VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING


---

Software Engineering Assignment

# Urban Waste Collection aid - UWC 2.0

**Semester: 222 - Group: HCMUT TheFinalists**

---

| | | |
|---|---|---|
| **Advisors:** | Bui Hoai Thang | |
| | Bui Cong Tuan | |
| | Nguyen Duc Anh | |
| | Quan Thanh Tho | |
| **Student:** | Vo Hoang Nhat Khang | 2152646 |
| | Le Trong Hieu | 2153342 |
| | Pham Duc Hai | 2052975 |
| | Le Hoang Mai Phuong | 2053349 |
| | Van Ngoc Thanh Tung | 2052782 |

HO CHI MINH CITY, OCTOBER 2022

# Contents

# Members list & Workload

| No. | Full name | Student ID | Duty in the assignment | Percentage of work |
|---|---|---|---|---|
| 1 | Vo Hoang Nhat Khang | 2152646 | Designing system, diagrams, wireframes, frontend and backend | 100% |
| 2 | Le Trong Hieu | 2153342 | Frontend and backend | 100% |
| 3 | Pham Duc Hai | 2052975 | Frontend | 100% |
| 4 | Le Hoang Mai Phuong | 2053349 | Frontend | 100% |
| 5 | Van Ngoc Thanh Tung | 2052782 | Frontend | 100% |

# Part I

# Problem specification

Urban waste management is one of several significant problems faced by many countries in the world and thus considered one of the important points to be improved in Sustainable Development Goal (SDG) 11: Sustainable cities and communities and SDG 6: Clean water and sanitation. Particular attention is given to developing countries that continue to prioritize development and economic growth. In urban context, solid waste management is costly and ineffective. Improvement of waste collection and management is emphasized by governments and organizations for positive impacts on cities, societies and environments.

Waste collection is often designated to an organization that provides professional waste management services. A typical waste collection process involves **(1)** back officers, who operate a central system to create calendar, coordinate front collectors and janitors, **(2)** collectors, who drive different types of vehicles and **(3)** janitors who manually collect garbage from Major Collecting Points (MCPs). Calendar and tasks were assigned among teams of janitors and coordinated by back officers. These assignments are often arranged in a weekly basic. Back officers also plan which vehicles to use and their routes. This planning activity happens every month. Everyday, the back officers sent messages with information about collecting route and time to collectors and janitors. Janitors use trollers (see Figure 1b) to collect garbage in their assigned areas and deliver to the MCPs. Collectors will pick up garbage from all janitors at an MCP. One collector drives only one vehicle during his working shift. The collector will drive through several MCPs with a predetermined route by back officers.


Figure 1a. Collecting vehicles

Figure 1b. Trollers

Organization X is contracted to develop an information management system called UWC 2.0 in order to improve efficiency of garbage collection of Service provider Y. The solution will include a Task Management module that allows back officers to:

1. Have an overview of janitors and collectors, their work calendar
2. Have an overview of vehicles and their technical details (weight, capacity, fuel consumptions, etc)
3. Have an overview of all MCPs and information about their capacity. Information should be updated from MCPs every 15 minutes with the availability of at least 95% of their operating time.
4. Assign vehicles to janitors and collectors
5. Assign janitors and collectors to MCPs (task)
6. Create a route for each collector. Assigned route is optimized in term of fuel consumption and travel distance.
7. Be able to send message to collectors and janitors

Also, the Task Management module will also allow the collectors and janitors to:

1. Have an overview of their work calendar
2. Have a detail view of their task on a daily and weekly basic. All important information should be displayed in one view (without scrolling down).
3. Be able to communicate with collectors, other janitors and back officers. The messages should be communicated in a real-time manner with delay less than 1 second.
4. Check in / check out task every day
5. Be notified about the MCPs if they are fully loaded

There are some constraints to the development of UWC 2.0. There exists a current system UWC 1.0 with a database. UWC 2.0 is expected to import and to use the existing data from UWC1.0. It is expected that the Task Management to be inter-operable with the UWC 1.0 as much as possible. The system should be able to handle real-time data from at least 1000 MCPs at the moment and 10.000 MCPs in five years. UWC 2.0 system interfaces should be in Vietnamese, with an opportunity to switch to English in the future.

# Part II

# Task solution

# Chapter 1

# Task 1: Requirement elicitation

**1.1 Describe the domain context of Urban waste management in Vietnam. Who are relevant stakeholders? What are their current needs? What could be their current problems? In your opinion, what benefits UWC 2.0 will be for each stakeholder?**

### 1.1.1 Describe the domain context of Urban waste management in Vietnam

As an emerging industrial hub with accelerating economic growth, Vietnam has severe environmental issues, particularly in waste management and plastic pollution. The total volume of waste each year in the country is approximately about 25.5 million tons, of which 75 percent goes into landfills.

The common routine for waste collection around the world nowadays usually involves back officers - planning and coordinating janitors and collectors; janitors - collecting waste and moving them to MCPs, and collectors - driving collecting vehicles to MCPs to collect waste. However, in Vietnam the routine is usually cut as collectors will manually collect waste from each household or a smaller scale of MCP. This causes the vehicle to cover almost every path available to pick up waste, resulting in traffic jams, time-wasting and possibly spreading the bad odor around the city. Therefore we urgently need to fix this bad routine as soon as possible.

To tackle the situation and towards a sustainable economy, the government has been deploying long-term action plans and setting ambitious recycling targets. Vietnam has launched a national action plan for the management of marine plastic litter, aiming to reduce 75 percent of Vietnam's marine plastic debris by 2030. By then, the country strives to eliminate the use of single-use plastics and non-biodegradable plastic bags from all coastal tourism areas. Meanwhile, all protected marine areas should be free of plastic litter. As a result, governments and businesses in Vietnam are focusing on mitigating the harmful effects of industrialization on the city, society and the environment.

**Figure 1.1:** The construction site of Thien Y waste-to-energy plant in Hanoi

### 1.1.2 Who are relevant stakeholders?

**Definition of stakeholder**

**A stakeholder** is a party that has an interest in a company and can either affect or be affected by the business. The primary stakeholders in a typical corporation are its investors, employees, customers, and suppliers. Nowadays, the concept has been extended to include communities, governments, and trade associations.

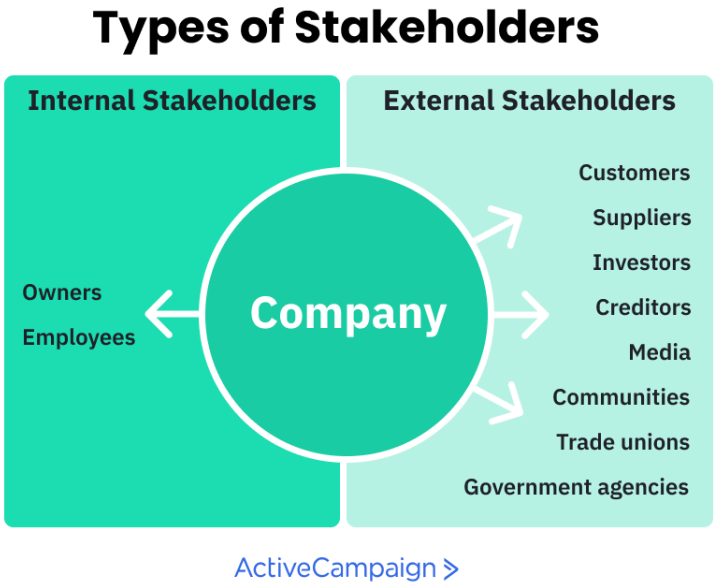



**Figure 1.2:** Stakeholder overview
**Source:** https://www.quora.com/Who-are-stakeholders

An entity's stakeholders can be both **internal** or **external** to the organization. Therefore, within the context of UWC 2.0, we decided to have the following categories:

- **Internal stakeholders:**
  - **Organization X**: Because X is contracted to develop the UWC 2.0 information management system.
  - **Service Provider Y**: As Y will receive the "software" from X and start deploying it on the city. In other words, Y plays as a base deploying role for the UWC 2.0 project.
  - **Back officers**: They manage calendars as well as creating plans for collectors and janitors.
  - **Collectors**: They work with janitors to collect the waste.
  - **Janitors**: The ones who directly collect the waste of the city.
- **External stakeholders:**
  - **Government:** The government are trying to propose a solution for reducing the severe effect of industrialization on the surrounding environment.
  - **Society:** UWC 2.0 is created for assisting the purpose of improving the waste-collect process, which in fact will have a decent on the people living in the city.

### 1.1.3 What are their current needs?

The needs may varies, based on different stakeholders. But overall:

- **Organization X:** They may need resources as well as develop ideas for the UWC 2.0 project. For instances, they may need a developer team, a project manager and some collaborators...

- **Service provider Y:** Clearly, UWC 2.0 is an improved version that can use existing data from UWC 1.0. Furthermore, new system can effectively manage resources. A good improvement maybe adding a language translation so that the app can be used in any language in any country, in this specific case is Vietnamese
- **Back officers:**
  1. Have an overview of janitors and collectors, their work calendar
  2. Have an overview of vehicles and their technical details (weight, capacity, fuel consumptions, etc)
  3. Have an overview of all MCPs and information about their capacity. Information should be updated from MCPs every 15 minutes with the availability of at least 95% of their operating time.
  4. Assign vehicles to janitors and collectors
  5. Assign janitors and collectors to MCPs (task)
  6. Create a route for each collector. Assigned route is optimized in term of fuel consumption and travel distance.
  7. Be able to send message to collectors and janitors
- **Collectors and janitors:**
  1. Have an overview of their work calendar
  2. Have a detail view of their task on a daily and weekly basic. All important information should be displayed in one view (without scrolling down).
  3. Be able to communicate with collectors, other janitors and back officers. The messages should be communicated in a real-time manner with delay less than 1 second.
  4. Check in / check out task every day
  5. Be notified about the MCPs if they are fully loaded
- **Government:** They need a better way of handling trash and contaminated objects from polluting the living area.
- **Society:** Everyone have the rights to live in a less polluted area. Urban citizens are in dead needing a better method of managing those trash, so that the environment can be protected and cleaner. Trash can then be classified better, which is more useful for future needs.

### 1.1.4 What could be their current problems?

- **Organization X:** In this scenario, X maybe lack of resources and they may need another organization to help deploying the software. Besides, it is pretty much unclear that the first and second version of the software is compatible or not.
- **Service provider Y:** Having to manually decide with route is optimal for the collectors. Coordinating units are difficult to manage. The first version of the software may not provide enough features as well as its scalability.
- **Back officers:** They maybe overwhelmed by how much information they need to control. There is a huge amount of resource to manage. Furthermore, the first version may not have the feature of keeping up the technical details such as weight, loading capacity or date/time... which makes it really hard to stay up-to-date. The message feature may also be missing at the first version, where officers are not able to communicate or directing the janitors/collectors heading to the right location.
- **Collectors and janitors:** It is a necessity to integrate more useful features into a single application, not separating the features which in fact makes the usage become confused and not appetizing. They are the ones who need calendar and maps to know exactly what they are doing and where to go. Message feature is a good idea here, basing on the fact that the system are organized like a network. People connect and work together as a team, and this comes down to the message feature that supports team communication more effectively. In some country, there are not many available software assisting this issue, which reduce the effectiveness as well as productivity of the company.
- **Government:** There are several possible problem, such as the cost of maintenance is not affordable. Or the first version of UWC is not able to deal with the raising industrialization and population growth, which badly affect the surrounding environment and therefore violates the regulations of SDGs.

- **Society:** Polluted areas are having a severe effect on people's life. The traditional method of managing waste is no longer suitable with today's living standard and need a better approach. Figure out how to handle trash and categorizing them into groups is one of the most important tasks nowadays, which not only correctly process those which maybe harmful to the living areas, but also open up more living space and kind of make trash reusable.

**Figure 1.3:** Garbage collect
**Source:** https://www.freepik.com/premium-vector/garbage-collection-waste-recycling-transportation-city_21655648.htm

### 1.1.5 In your opinion, what benefits UWC 2.0 will be for each stakeholder?

For specific stakeholders, it is expected that they will receive different benefits based on their need if the UWC 2.0 is deployed and got approved. To be more specific:

- **Organization X:** Better way of handling those waste is a great opportunity to focus more on their further development and economic raise. They can also earn some profit and some reputation as well. Or they broaden their partnership network by connecting with service provider Y, which is a good thing if they tend to develop the app or having upcoming projects.
- **Service provider Y:** Improve the process of collecting waste, garbage, increase profit with expanding the scope of collection routes. They can gain some profit as well, and it is pretty much clear that if UWC 2.0 can utilize the original features and add some more new useful ones, they can better maintenance and control the application better.
- **Back officers:** Having a better overview of vehicles, their team work progress, and the most important is to assigning task and assisting their team in time. This can help organizing the work more efficient. Not only this can reduce the cost of fuel consumption for each time of collecting waste but this also makes the process easier to understand and to execute. The fact that message feature comes into play will be perfect as now everything will get its flow.
- **Collectors and janitors:** The UWC 2.0 is a great tool for the collectors and janitors to control their working flow: From calendars, working ships, maps, to the message or calling for assistance.
- **Government:** With UWC 2.0 it is now possible for them to satisfy the requirements of the SDGs, further extending the benefits of not only the people of living areas but also the companies taking part in this project.
- **Society:** They can live in a fresher atmosphere and they can also take part in handling domestic waste to keep the area as clean as possible.

**Figure 1.4:** Collecting trash on the street
**Source:** https://connect2local.com/l/610324/c/972113/3-benefits-of-routine-garbage-collection-services

## 1.2 Describe all functional and non-functional requirements that can be inferred from the project description. Draw a general use-case diagram for the whole system

Requirements that are well thought through and clearly documented are essential to any successful software engineering project. There are two main different types of system requirements that should be gathered by those working on software projects. System requirements can be categorized as either functional requirements or non functional requirements.

### 1.2.1 Describe all functional and non-functional requirements that can be inferred from the project description

**Functional requirements and non-functional requirements**

- **Functional requirements:** Specific features and functions that a system must possess in order to meet the needs and expectations of its stakeholders. They define what the system must do and are usually described in terms of inputs, processes, and outputs.
- **Non-functional requirements:** Specific attributes or qualities that a system must possess, but they are not directly related to a particular function or feature of the system.

To make it easier to control the requirements as well as proposing them later on whenever the whole system has any problems, we will assign the requirements with a specific **ID**. So, the requirements table will have the form <**ID**> - <**Problem**>

1. **Functional requirements:**
   - **About the system:**

| ID | Requirements/Description |
|---|---|
| F-SYSTEM-0 | Register page and login page for users |
| F-SYSTEM-1 | Access restriction based on role of user |
| F-SYSTEM-2 | Be able to import and use data of UWC 1.0 |
| F-SYSTEM-3 | Have a map showing locations of vehicles as well as MCPs |

   - **About the collectors and janitors:**

| ID | Requirements/Description |
|---|---|
| F-WORKER-0 | Be able to view worker's profile |
| F-WORKER-1 | Be able to view work calendar and tasks of each day shown on the calendar |
| F-WORKER-2 | Be able to view weekly tasks |
| F-WORKER-3 | Be able to view check in / check out tasks |
| F-WORKER-4 | Be able to view MCPs location |
| F-WORKER-5 | They can be notified about the MCPs that are fully loaded on the map |
| F-WORKER-6 | Be able to message co-workers or back officers |
| F-WORKER-7 | View overall map |

- **About the back officers:**

| ID | Requirements/Description |
|---|---|
| F-OFFICER-0 | Have a dashboard linking to other sections and include summarized data, such as map, working staffs, overloaded MCPs,... |
| F-OFFICER-1 | Be able to monitor all MCPs and vehicles via the map, as well as notice overloaded MCPs by marking them. |
| F-OFFICER-2 | Be able to view personal information of workers |
| F-OFFICER-3 | Be able to view workers' work calendar |
| F-OFFICER-4 | Be able to view vehicles and their technical details (weight, capacity, fuel consumption, etc). |
| F-OFFICER-5 | Be able to have an overview of all MCPs and information about their capacity |
| F-OFFICER-6 | Be able to assign vehicles to workers. |
| F-OFFICER-7 | Be able to assign MCPs to workers. |
| F-OFFICER-8 | Be able to create an optimized route (in terms of distance and fuel, based on assigned MCPs) for each collector. |
| F-OFFICER-9 | Be able to send messages to collectors and janitors (maybe for further instructions on the task). |

2. **Non-functional requirements:**

- **About the usability:**

| ID | Requirements/Description |
|---|---|
| NF-USAGE-0 | The system should be available on macOS and Windows |
| NF-USAGE-1 | The system should be available in both English and Vietnamese |
| NF-USAGE-2 | The users should be able to use to the app effectively after 1-2 hours of training |
| NF-USAGE-3 | Important information (janitor and collector's Task Management) should be displayed in one view |

- **About the space:**

| ID | Requirements/Description |
|---|---|
| NF-SPACE-0 | The server can handle at most 2GB data at a time. |

- **About the performance:**

| ID | Requirements/Description |
|---|---|
| NF-PERFORMANCE-0 | Messages should be communicated in real-time manner, delay less than 1 second |
| NF-PERFORMANCE-1 | Response time when users interact with the system should be less than 1 second. |

- **About the dependencies:**

| ID | Requirements/Description |
|---|---|
| NF-DEPEND-0 | The system should be available during working time: 9am-5pm. |
| NF-DEPEND-1 | The system should have its database backup |
| NF-DEPEND-2 | System's failure rate should be less than 2% |

- **About the operations:**

| ID | Requirements/Description |
|---|---|
| NF-OPERATION-0 | The system should be able to handle real-time data from at least 1000 MCPs at the moment and 10.000 MCPs in five years. |
| NF-OPERATION-1 | Information about MCPs' capacities should be updated every 15 minutes |
| NF-OPERATION-2 | Route planning, MCPs and vehicle assignment should be done on the 29th of every month. |

- **About the security:**

| ID | Requirements/Description |
|---|---|
| NF-SECURITY-0 | The system should satisfy basic security's needs. |

- **About the development:**

| ID | Requirements/Description |
|---|---|
| NF-DEVELOPMENT-0 | The system should be written in JavaScript (ReactJS and NodeJS). |
| NF-DEVELOPMENT-1 | The backend of the system should be connected to `MongoDB`. |

### 1.2.2 Draw a general use-case diagram for the whole system

**UML use-case diagram**

Use-case diagrams are usually referred to as **behavior diagrams**, used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.
Use case diagrams are used to specify:

- (external) requirements, required usages of a system under design or analysis (subject) - to capture what the system is supposed to do
- the functionality offered by a subject – what the system can do
- requirements the specified subject poses on its environment - by defining how environment should interact with the subject so that it will be able to perform its services.

In system use-case diagram, there are some typical components:

- **Actor:** a graphical representation of a person, system, or external entity that interacts with a system to achieve a specific goal. In our project, there are 3 main actors: Back officer, Janitor and Collector
- **Use case:** A set of actions, services, and functions that a system provides to its users. A use-case defines a goal that a user or system has with the system being modeled, and it describes the steps involved in achieving that goal. Basic use-cases are defined quite clearly in the diagram below, including Manage tasks, assigning tasks, to sending out messages, log in, log out,...
- **Communication links:** Represents a relationship between two instances in a system. It represents the flow of information or data between the instances, and it can be used to model the interactions between objects, components, or actors in a system.
- **Boundary of system:** Represents the external limits of the system, separating it from its environment. It defines the interface between the system and the actors that interact with it, and it specifies what the system does and does not do. So, quite clearly that Authentication (Log in / Log out module) and Task assignment module are 2 main Boundary in the system.

- **Relationships:** Used to describe the connections or associations between elements in a system, such as classes, objects, use-cases, actors, components, and so on.

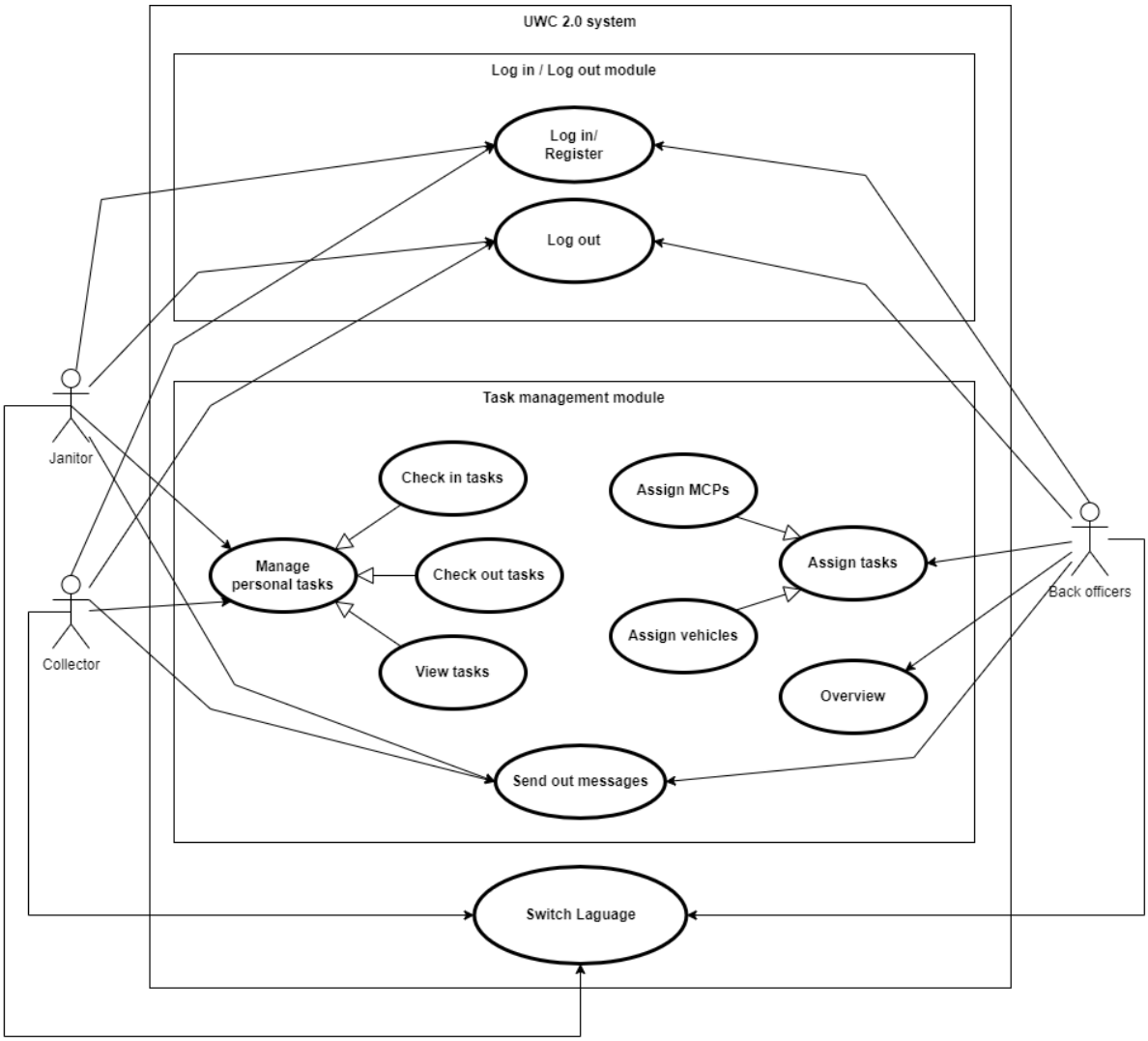



**Figure 1.5:** Use-case diagram for the whole system

## 1.3 For the Task assignment module, draw its use-case diagram and describe the use-case using a table format

### 1.3.1 Draw Task assignment use-case diagram

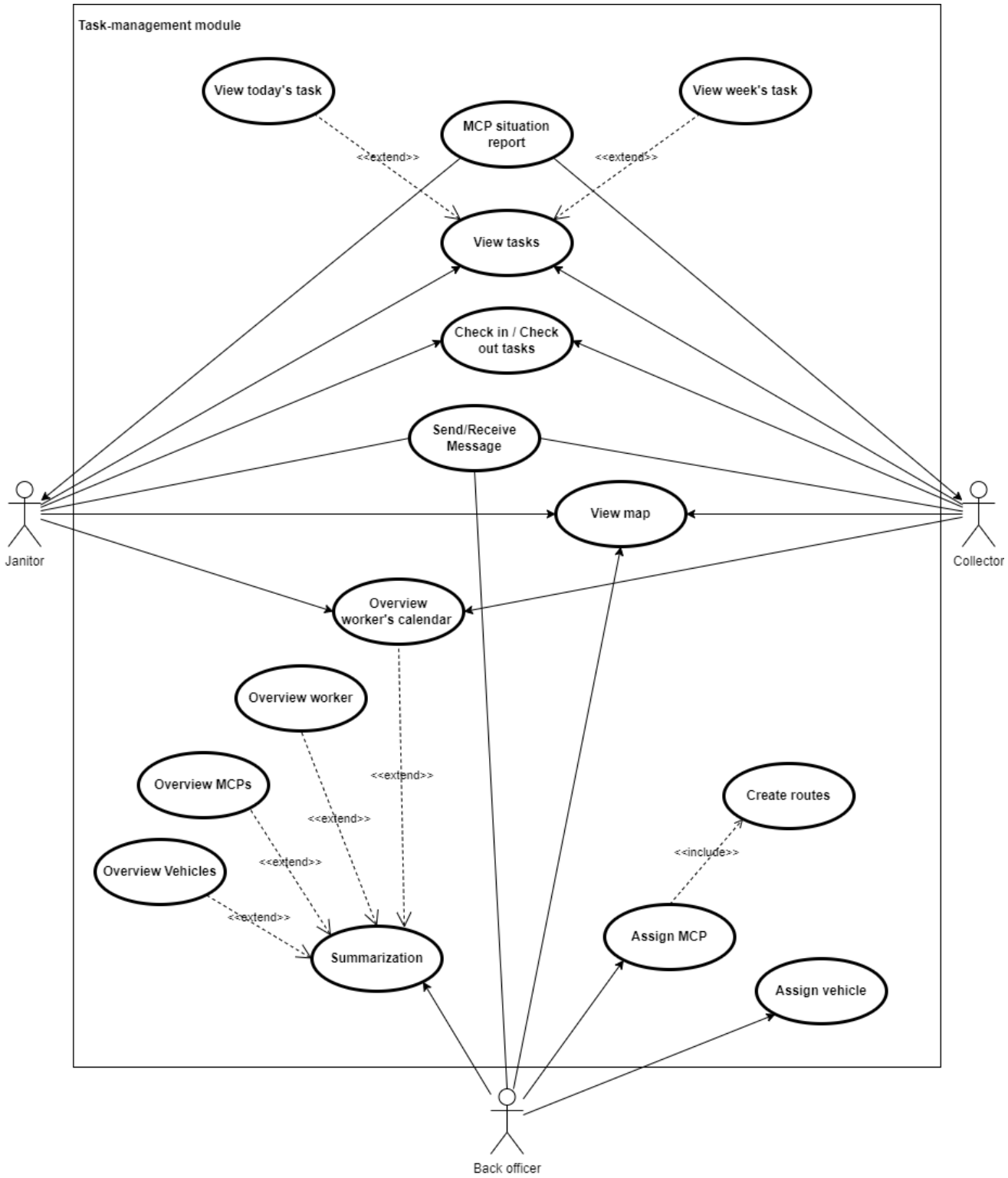



**Figure 1.6:** Use-case diagram for task assignment system

### 1.3.2 Describe the use-case using a table format

#### 1.3.2.1 Summarizing of vehicles

| | |
|---|---|
| **ID and Name** | F-OFFICER-3 Overview vehicles |
| **Actor** | Back officers |
| **Description** | Get vehicles' technical details |
| **Trigger** | User will click on the "**Vehicle overview**" tab designed on the sidebar |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Back officers is logged in |
| **Post-conditions** | Back officers receive the details of all vehicles on the screen |
| **Normal flow** | 1. The back officer provides login information<br>2. The system verifies the identity of the back officer<br>3. The system will redirect the user to the dashboard view<br>4. The back officer views the vehicles' technical details by selecting the "**Vehicles**" tab on the sidebar. |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

#### 1.3.2.2 View collectors/janitors' information

| | |
|---|---|
| **ID and Name** | F-OFFICER-2 Overview worker |
| **Actor** | Back officer |
| **Description** | Get an overview of janitors and collectors personal information |
| **Trigger** | Click on the icon on the sidebar belonging to the worker. |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Back officers is logged in |
| **Post-conditions** | Receive all information of the collectors and janitors he/she is interested in |
| **Normal flow** | 1. The actor provides login information<br>2. The system verifies the identity of the actor<br>3. The system will redirect the user to the dashboard view<br>4. The actor views the workers' information by selecting the icon on the sidebar belonging to the worker. |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

#### 1.3.2.3 View collectors/janitors' work calendar

| | |
|---|---|
| **ID and Name** | F-OFFICER-3 Overview workers' calendar |
| **Actor** | Back officers, collectors, janitors |
| **Description** | Provide information about collectors and janitors' schedule. |
| **Trigger** | User press the "**Detailed schedule**" button |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. User is logged in |
| **Post-conditions** | The worker's calendar is displayed |
| **Normal flow** | 1. The back officer provides login information<br>2. The system verifies the identity of the back officer<br>3. The system will redirect the user to the dashboard view<br>3.1. For personal view, user can view personal calendar by going to his/her profile and select "**Calendar**" tab.<br>3.2. In the case that the back officer views the workers' schedule, they will select the "**Calendar**" button on the sidebar of the worker's profile. |
| **Alternative flow** | Workers are not allowed to see each others' schedule. Which means that in the view of workers, they will not be able to see the "**Calendar**" button. |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

#### 1.3.2.4 Summarize MCPs

| | |
|---|---|
| **ID and Name** | F-OFFICER-5 Overview MCPs |
| **Actor** | Back officer |
| **Description** | Get an overview of MCPs and their current status |
| **Trigger** | Users click on the "**MCPs Overview**" button on the sidebar |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Back officer is logged in |
| **Post-conditions** | Back officer receives all MCPs information on the screen |
| **Normal flow** | 1. The back officer provides login information<br>2. The system verifies the identity of the back officer<br>3. The system will redirect the user to the dashboard view<br>4. The back officer views the summarized MCPs information by clicking the "**MCPs Overview**" on the left sidebar. If back officers want to know the location of MCPs, they can click the "**MCP**" tab and click on the re-center icons to view them live. |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

#### 1.3.2.5 Assign MCPs

| | |
|---|---|
| **ID and Name** | F-OFFICER-7 Assign MCP |
| **Actor** | Back officer |
| **Description** | Back officers can assign MCPs to workers |
| **Trigger** | Back officer access to the "**Task assignment**" on the sidebar, and click "**Assign MCP(s) for workers**" button. |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Back officer is logged in and access to the "**Task assignment**" on the sidebar. |
| **Post-conditions** | MCPs are assigned to workers afterwards |
| **Normal flow** | 1. The officer click on the "**Task assignment**" on the sidebar<br>2. They assign MCPs to workers (by changing the name that is in charge for a specific MCP).<br>3. Back officer click "**Assign MCP(s) for workers**" button and a dialog appears on the screen for user to make them choose workers in charge for that MCP. |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password<br>At the assigning step (**step 3**): In the case of unsuccessful assignment, a message appears on the screen "**Cannot assign**" (MCP already taken by another worker) |

#### 1.3.2.6 Assign vehicle

| | |
|---|---|
| **ID and Name** | F-OFFICER-6 Assign vehicle |
| **Actor** | Back officer |
| **Description** | Assign vehicles to the workers |
| **Trigger** | Back officer access to the "**Vehicle assignment**" on the sidebar, choose the MCP, and then click "**Assign vehicle**" button. |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Back officer is logged in and access to the "**Vehicle assignment**" on the sidebar. |
| **Post-conditions** | Vehicles are assigned to workers |
| **Normal flow** | 1. The officer click on the "**Vehicle assignment**" on the sidebar<br>2. They choose MCP to assign vehicle.<br>3. Back officer click "**Assign now**" button and a dialog appears on the screen, making the user to choose the which truck to assign. |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password<br>At the assigning step (**step 3**): In the case of unsuccessful assignment, a message appears on the screen "**Cannot assign**" (maybe because unavailable vehicle at that particular moment) |

#### 1.3.2.7 Create route

| | |
|---|---|
| **ID and Name** | F-OFFICER-8 Create routes |
| **Actor** | Back officer |
| **Description** | Back officer can create an optimized route (in terms of distance and fuel, based on assigned MCPs) for each collector. |
| **Trigger** | Back officer clicks the "**Route Design**", draw out a set of consecutive routes from the workers' location to the required MCP, then click "**Route confirm**" button. A dialog will appears "**Route implemented**" |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Back officer is logged in and access to the "**Route Design**" on the sidebar. |
| **Post-conditions** | Route is assigned to worker |
| **Normal flow** | 1. The officer click on the "**Route Design**" on the sidebar<br>2. They draw out a set of consecutive routes from the workers' location to the required MCP.<br>3. Back officer click "**Route confirm**" button and a dialog appears on the screen "**Route implemented**" |
| **Alternative flow** | When the route is unavailable, a message is displayed "**Route unavailable**" |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password<br>At **step 3**, if route cannot be established (cannot form a complete path to the MCP) then there's a message on the screen: "**Route is invalid**" |

#### 1.3.2.8 Send/Receive Message

| | |
|---|---|
| **ID and Name** | (F-OFFICER-9 + F-WORKER-6) Send/Receive Message |
| **Actor** | Back officer, Janitor and Collector |
| **Description** | Back officer, Janitor and Collector can communicate to each other by message feature |
| **Trigger** | User clicks on the "**Message**" tab on the navigation bar |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. User is logged in and access to the "**Message**" on the sidebar. |
| **Post-conditions** | Message is sent via the feature |
| **Normal flow** | 1. The actor provides login information<br>2. The system verifies the identity of the actor<br>3. The system will redirect the user to the dashboard view<br>4. The user click on the "**Message**" on the sidebar<br>5. Now can start choosing individuals or group they want to communicate and start chatting |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password<br>- When the message sending procedure is interrupted (due to weak Internet connection,. . . ), a message is displayed to inform the user about the incident and suggest them to try again "**Please try again later**". |

#### 1.3.2.9 View map

| | |
|---|---|
| **ID and Name** | F-SYSTEM-3 View map |
| **Actor** | Back officer, Collector and Janitor |
| **Description** | Back officer, Collector and Janitor can view the map |
| **Trigger** | After user login into the system, they can see the map |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute |
| **Post-conditions** | The users can view a general map containing MCPs location and other workers' information |
| **Normal flow** | 1. The actor provides login information<br>2. The system verifies the identity of the actor<br>3. The system will redirect the user to the dashboard view containing the map |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

#### 1.3.2.10 Check in/Check out task

| | |
|---|---|
| **ID and Name** | F-WORKER-3 Check in / Check out task |
| **Actor** | Collector and Janitor |
| **Description** | Collector and Janitor are able to view check in / check out tasks |
| **Trigger** | The worker click "**I got it**" (Check in) or "**Finished**" (Check out) button next to the task |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Collector and Janitor are logged in |
| **Post-conditions** | 1. Janitor can know whether the collector finished a task<br>2. Back officer can know whether the worker finished the required tasks |
| **Normal flow** | 1. The worker will click on the "View task" button on the sidebar. They can view the task of that day and start working on it by clicking "**I got it**" button next to the task. This is called "**Check in**" task. Worker's status changes to "**Working**"<br>2. After finishing the work, the worker can click on the button "**Finished**" for button for checking out next to the task. Worker's status now changes to "**Done**" |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

#### 1.3.2.11 View task

| | |
|---|---|
| **ID and Name** | F-WORKER-2 View task |
| **Actor** | Collector and Janitor |
| **Description** | Collector and Janitor can view their daily and weekly task |
| **Trigger** | Collector and Janitor click the "**View task**" button on the sidebar |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Collector and Janitor are logged in |
| **Post-conditions** | Collector and Janitor can check their daily and weekly task |
| **Normal flow** | 1. The actor provides login information<br>2. The system verifies the identity of the actor<br>3. The system will redirect the user to the dashboard<br>4. The Collector and Janitor click the "**View task**" button on the sidebar |
| **Alternative flow** | When there is no task, a message is displayed "**No task assigned**". |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

#### 1.3.2.12 MCP situation's report

| | |
|---|---|
| **ID and Name** | F-OFFICER-7 Assign MCP |
| **Actor** | Back officer |
| **Description** | Back officers can assign MCPs to workers |
| **Trigger** | Back officer access to the "**Task assignment**" on the sidebar. |
| **Pre-conditions** | 1. The system and database are available<br>2. The internet is available<br>3. The website has been implemented and ready to execute<br>4. Back officer is logged in and access to the "**Task assignment**" on the sidebar. |
| **Post-conditions** | MCPs are assigned to workers afterwards |
| **Normal flow** | 1. The officer click on the "**Task assignment**" on the sidebar<br>2. Below each bar of the MCPs, the Back Officer can see MCPs status. |
| **Alternative flow** | None |
| **Exceptions** | At the verification step (**step 2**):<br>- Registering into the web:<br>+ Invalid email<br>+ New password is too short or too weak for security aspect<br>- Logging into the web:<br>+ Invalid email<br>+ User email doesn't exist in the database<br>+ Incorrect password |

# Chapter 2

# Task 2: System modelling

## 2.1 Draw an activity diagram to capture the business process between systems and the stakeholders in Task Assignment module

**Activity diagram**

The Unified Modeling Language includes several subsets of diagrams, including structure diagrams, interaction diagrams, and behavior diagrams. **Activity diagrams**, along with use case and state machine diagrams, are considered behavior diagrams because they describe what must happen in the system being modeled.
Activity diagrams help people on the business and development sides of an organization come together to understand the same process and behavior.
Some of the most common components of an activity diagram include:

- **Action:** A step in the activity wherein the users or software perform a given task. In Lucidchart, actions are symbolized with round-edged rectangles.
- **Decision node:** A conditional branch in the flow that is represented by a diamond. It includes a single input and two or more outputs.
- **Control flows:** Another name for the connectors that show the flow between steps in the diagram.
- **Start node:** Symbolizes the beginning of the activity. The start node is represented by a black circle.
- **End node:** Represents the final step in the activity. The end node is represented by an outlined black circle.

### 2.1.1 Diagram:

To view more clearly, follow this link: https://drive.google.com/file/d/1oFCEzmusOO1KySQDG-9M_Zp_NdsNva-d/view?usp=sharing

### 2.1.2 Description:

1. Users log in or register into the system
2. The message feature is available once the user is logged in. This feature will now run in **parallel** mode with the task-assignment module. It is because that meanwhile the worker are working on their task, they can still communicate to each other through the message system, or even the back officers can chat with the workers for the work.
3. Users can view their profile as well as a map at the beginning.
4. The main module works as follow:
    - **If the user is a worker (collector or janitor)**, he/she can check the daily task by clicking "**View task**" button. Now the system displays the task on the screen. After viewing the tasks, workers click

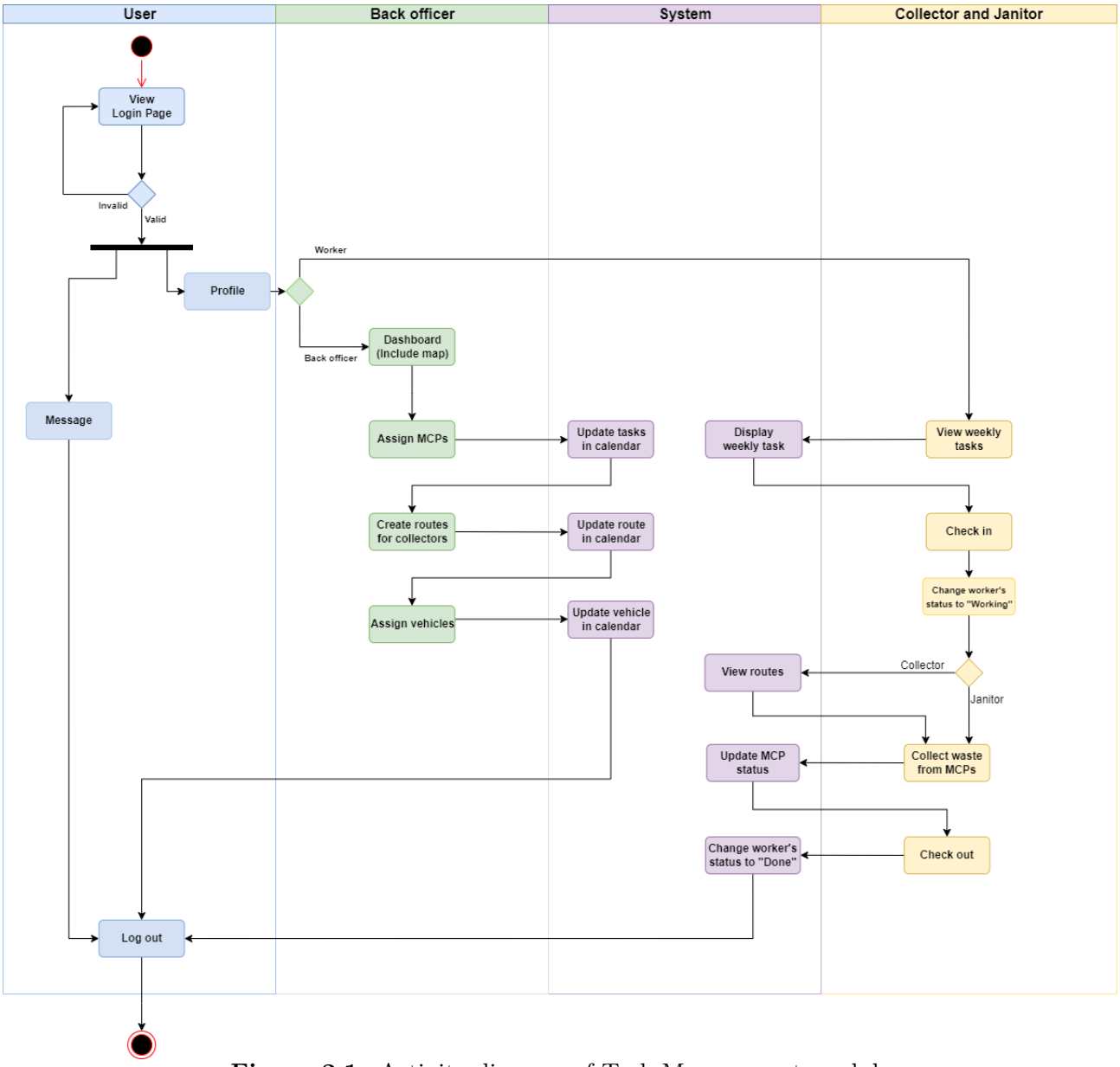



**Figure 2.1:** Activity diagram of Task Management module

on the "**Confirm**" button, and their status turn into "**Working**" on the system. The collector now view the routes assigned to them through the system, together with the janitor to collect waste from the MCPs. After finishing the tasks, workers click the button "**Finished**" next to the task. This changes to status to "**Done**" on the system.

- **If the user is a back officer**, he/she is now viewing the map and start assigning MCP as well as specific tasks to workers. After that, he/she creates routes to the determined MCP and the vehicle, respectively. Meanwhile, the system will automatically update the current information of the assigned fields.

5. If the user just want to login into the system to chat with the others then logging out, that's fine. Or, if they login in order to work, at the end of the day, they can just log out, and that's also fine.

**2.2 Think about a possible way for a back officer to assign vehicles to janitors and collectors. Draw a sequence diagram to visualize this process.**

### 2.2.1 Think about a possible way for a back officer to assign vehicles to janitors and collectors

1. The Back officer will choose MCP (assigning task) for the collectors.
2. After that, he/she will choose vehicle (trucks) for the collector. The status of the vehicle is also taken into account.
3. Back officers can also choose trolleys for janitors so that they can work at a specific MCP.

### 2.2.2 Draw a sequence diagram to visualize this process

**Sequence diagram**

**UML Sequence Diagrams** are interaction diagrams that detail how operations are carried out. They capture the interaction between objects in the context of a collaboration. Sequence Diagrams are time focus and they show the order of the interaction visually by using the vertical axis of the diagram to represent time what messages are sent and when. Sequence Diagrams captures:

- The interaction that takes place in a collaboration that either realizes a use case or an operation (instance diagrams or generic diagrams)
- High-level interactions between user of the system and the system, between the system and other systems, or between subsystems (sometimes known as system sequence diagrams)

### 2.2.3 Diagram

Follow this link for a better view: https://drive.google.com/file/d/17ZIrvTZr9oNB76NMxTv3yGIdzTepfXBA/view?usp=sharing

### 2.2.4 Description:

1. The officer start assigning process by clicking the "**Assign vehicle**" button.
2. Interface triggers the process by the `use()` function
3. The module will take charge interacting with the database to get the informations of the vehicles, workers and MCPs
4. Then, the module will return back the information to the interface and show the back officer.
5. While the back officer still assigning the vehicles:
   - 5.1 He/she choose a vehicle that he/she wants to assign
   - 5.2 Then the `assign()` function helps them in assigning vehicles to the workers. If a vehicle is validly assigned to the worker, a message is rendered on the screen "**Vehicle assigned**"
6. If the back officer wants to end the assigning process, he/she clicks the "**Done**" button. Now, the interface use the `updateAllVehicle()` to interact with the module. Finally is to update the database, and display full information.

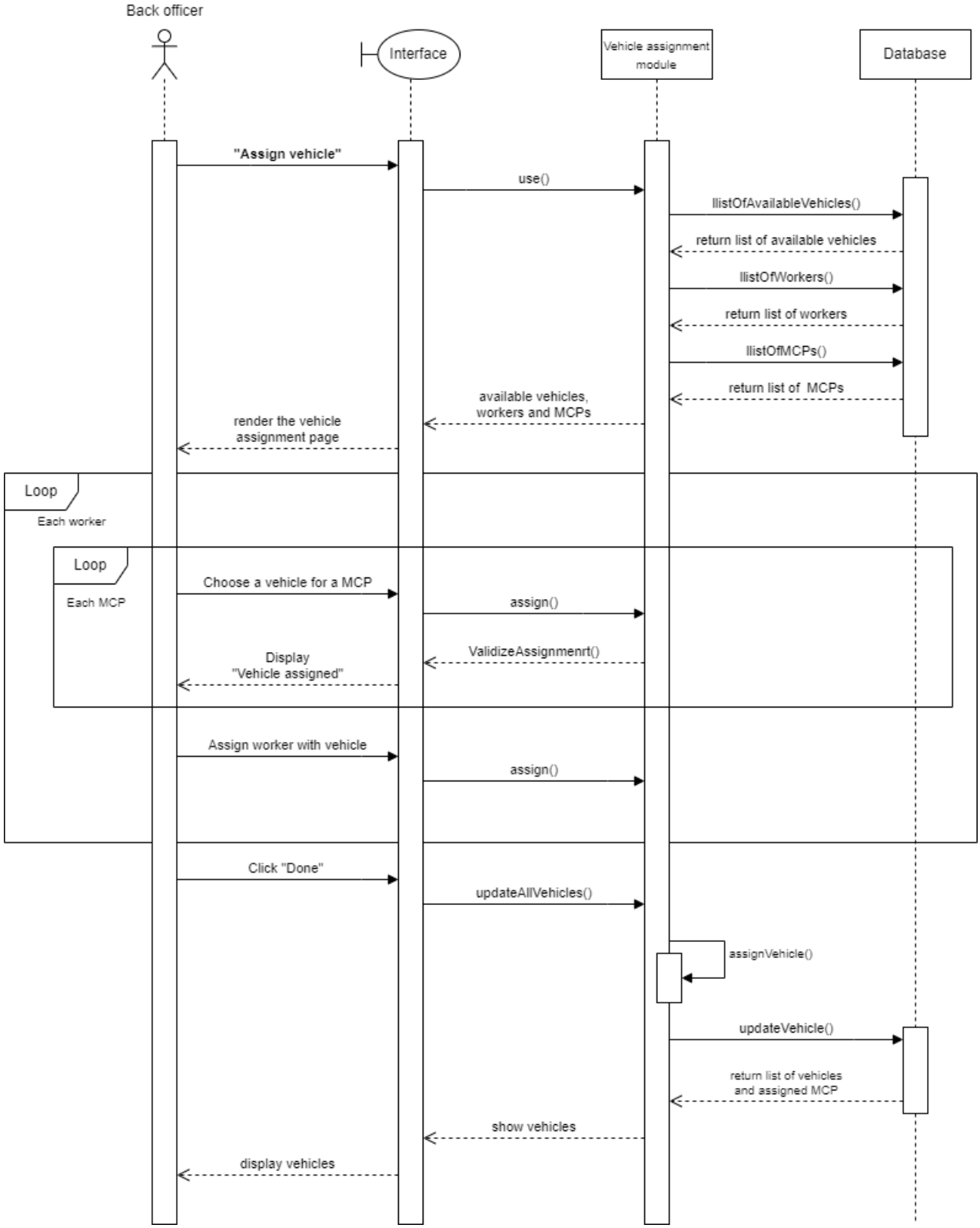



**Figure 2.2:** Sequence diagram how officers assign vehicles for workers

## 2.3 Draw a class diagram of Task Assignment module as comprehensive as possible

**Class diagram**

- In software engineering, a **class diagram** in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.
- The purpose of class diagram is to:
  1. Shows static structure of classifiers in a system
  2. Diagram provides a basic notation for other structure diagrams prescribed by UML
  3. Helpful for developers and other team members too
  4. Business Analysts can use class diagrams to model systems from a business perspective
- A UML class diagram is made up of:
  - A set of classes
  - A set of relationships between classes

### 2.3.1 Diagram

Follow this link to have a better view: https://drive.google.com/file/d/1EVexemJJrxhX195NPCHqQ2WEpR8NwqKF/view?usp=sharing

### 2.3.2 Description

For this diagram, we will observe it from 2 perspectives:

- **From top to bottom:** This approach provides us a look at the system layer. There are 5 main layers in the system:
  1. **Presentation layer:** The presentation layer, also known as the user interface or UI layer, is responsible for presenting data and information to the user in a way that is easy to understand and interact with. It may include components such as forms, menus, buttons, and other visual elements that enable the user to interact with the software. In the UWC 2.0, we decided to contain some basic UI such as:
     - Account management
     - Message
     - Task
     - MCP report
     - Vehicle
  2. **Business layer:** Also known as the domain layer or application layer, is where the core business logic of the application resides. It contains the application's rules and algorithms for processing and manipulating data, and often includes components such as workflows, business entities, and rules engines.
  3. **Service layer:** Provides a set of high-level interfaces or APIs that encapsulate the application's core business logic and functionality. It acts as a bridge between the presentation layer and the business layer, providing a well-defined interface that can be used by both.
  4. **Persistence layer:** responsible for storing and retrieving data from a database or other data storage system. It handles tasks such as creating, reading, updating, and deleting data, and often includes an object-relational mapping (ORM) framework to help map between the application's object model and the database's relational model.
  5. **Database layer:** Responsible for managing the application's data storage and retrieval needs. It includes components such as the database management system (DBMS), the database schema, and any middleware or drivers used to access the database. The database layer is responsible for ensuring

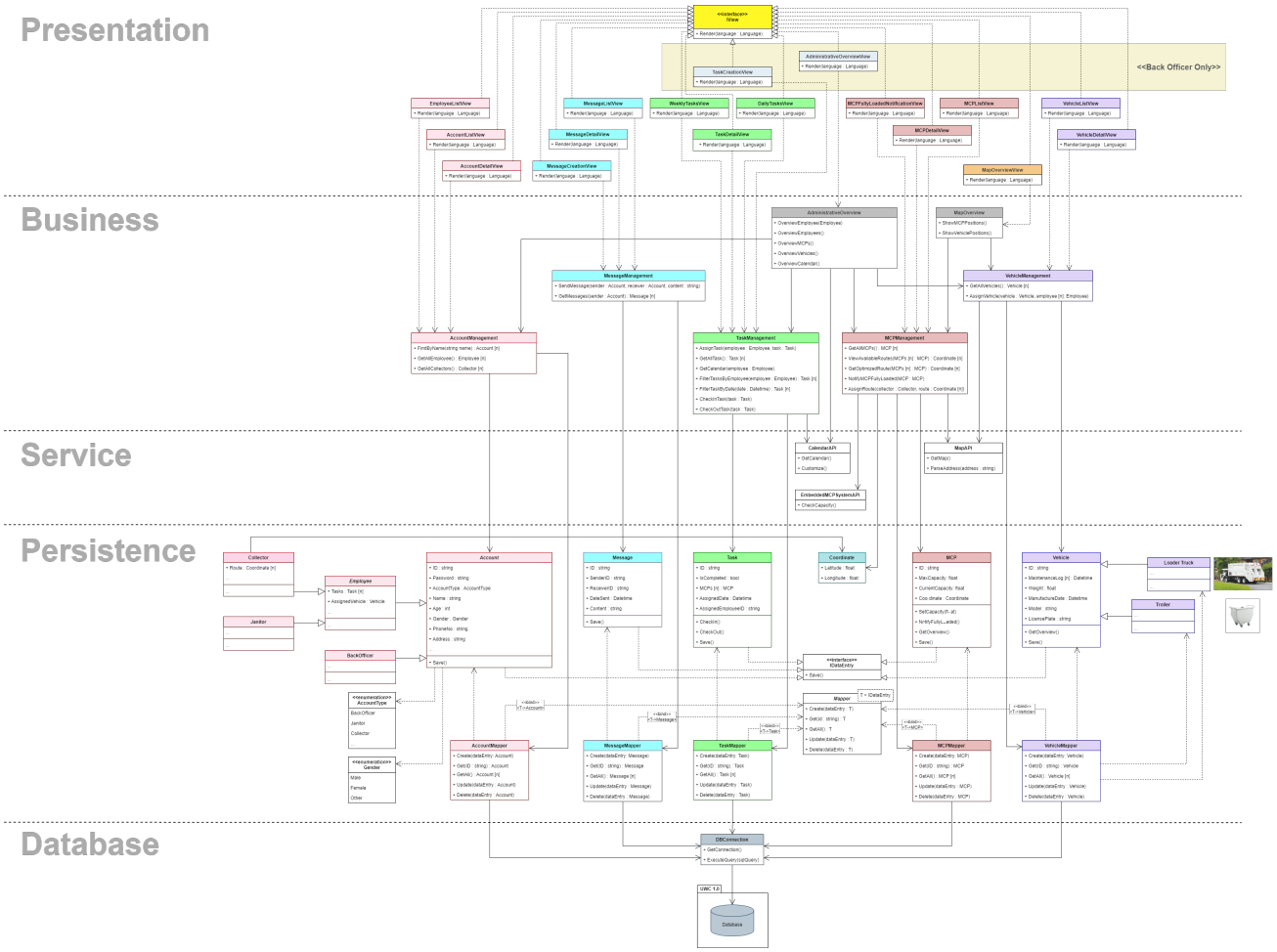



**Figure 2.3:** Class diagram of Task Assignment

data integrity, managing transactions, and providing efficient and reliable access to data for the other layers of the application. The database layer can be seen as a part of the persistence layer, as it handles the actual storage and retrieval of data from the database system.

- **From left to right:** This approach sketches out a general view of modules in the system. ‘

1. **Account system:** Managing UWC 2.0’s account. This system typically involves user authentication, user profile management, and account settings management. The process can be described as follow:
   1.1 The module receive related request from users’ account. It then passes the requests to appropriate components. In this case, `AccountManagement` will handle the requests based on its logic.
   1.2 When invoked, `AccountManagement` will send a request to `AccountMapper` to retrieve or update data in database.
   1.3 If retrieving data is the case, the data are then put in the Account class to be encapsulated in an Account object before sent back to `AccountManagement`. The data are then processed and displayed on the screen using relevant classes in presentation layer if necessary.
2. **Message system:** Responsible for processing messages (creating, sending, receiving, etc). Its managing mechanism is almost the same as that of account system except for one thing: Message class gets data from both `MessageMapper` and `Account` class (to indicate sender and receiver).
3. **Task system:** This system is specified for tasks handling and is similar in management process to message system. It also take advantage of calendar API to work with calendar. A task system is a system that allows users to create and manage tasks or to-do lists. This system typically involves task creation, task assignment, and task completion.
4. **Coordinate system:** In a trash collecting system, a coordinate system can be used to help identify the location of trash cans, garbage trucks, and other equipment used in the waste management process.

The coordinate system can be used to keep track of the location of different trash cans and to optimize routes for garbage trucks. Can also be used to store the coordinate of MCPs retrieved from the database and is used in MCP managing process.

5. **MCP system:** Responsible for working with MCPs' data. It managing mechanism is almost the same as that of account system except for one thing: `MCPManagement` class gets data from both `MCPMapper` and `Coordinate` class (to determine MCPs' location). It also uses APIs from service layer to interact with map and sensors in each MCP.
6. **Vehicle system:** Focuses on working with vehicles' data and has analogous mechanism to account system. It also uses map API to work with the map.
7. **Overview system:** Once it receives requests, it invokes corresponding management systems and APIs to get the overview data
8. **APIs:** This group includes `MapAPI`, `CalendarAPI` and `EmbededMCPSystemAPI`. These API provide the interface to work with external resources like map, calendar or sensors in MCPs.
9. **DBConnection:** This class plays a role as the middleman between persistence and database layer. It receives requests from mapper classes and provide appropriate services to work with the database. The purpose of this system is to establish a secure and efficient connection between the application and the database, allowing the application to retrieve, store, and manipulate data in the database.

## 2.4 Develop MVP 1 as a user interface of either a Desktop-view central dashboard for Task Management for back-officers OR a Mobile-view Task assignment for Janitors and Collectors. Decide yourself what to include in the view. Use a wireframe tool like Figma or Adobe XD, or Illustrator

**Figma link:** https://www.figma.com/proto/Uau2IZfLc3J35ByCCjvWg5/UWC-2.0?type=design&node-id=73-3&scaling=scale-down&page-id=0%3A1&starting-point-node-id=73%3A3

### 2.4.1 Log in/ Register

First, the user will start with logging into the system:


**Figure 2.4:** Logging role

In case the user is a new one, then they can click "**Register**" button at the bottom of the blackbox. A new window will appears.


**Figure 2.5:** Register page

After registering, now user provides his/her username and password.


**Figure 2.6:** Logging into the system

### 2.4.2 Central dashboard

After having successful login, user reaches the dashboard:


**Figure 2.7:** Dashboard

Dashboard is considered the homepage of our system. This contains viewing profile, messages, notifications, calendar, and assigning tasks, vehicles, viewing workers, viewing MCPs and setting. In this part, we will demonstrate how Back Officer can assign tasks to their workers.

### 2.4.3 Start assigning task

From dashboard, we click on the "**Assign**" button.

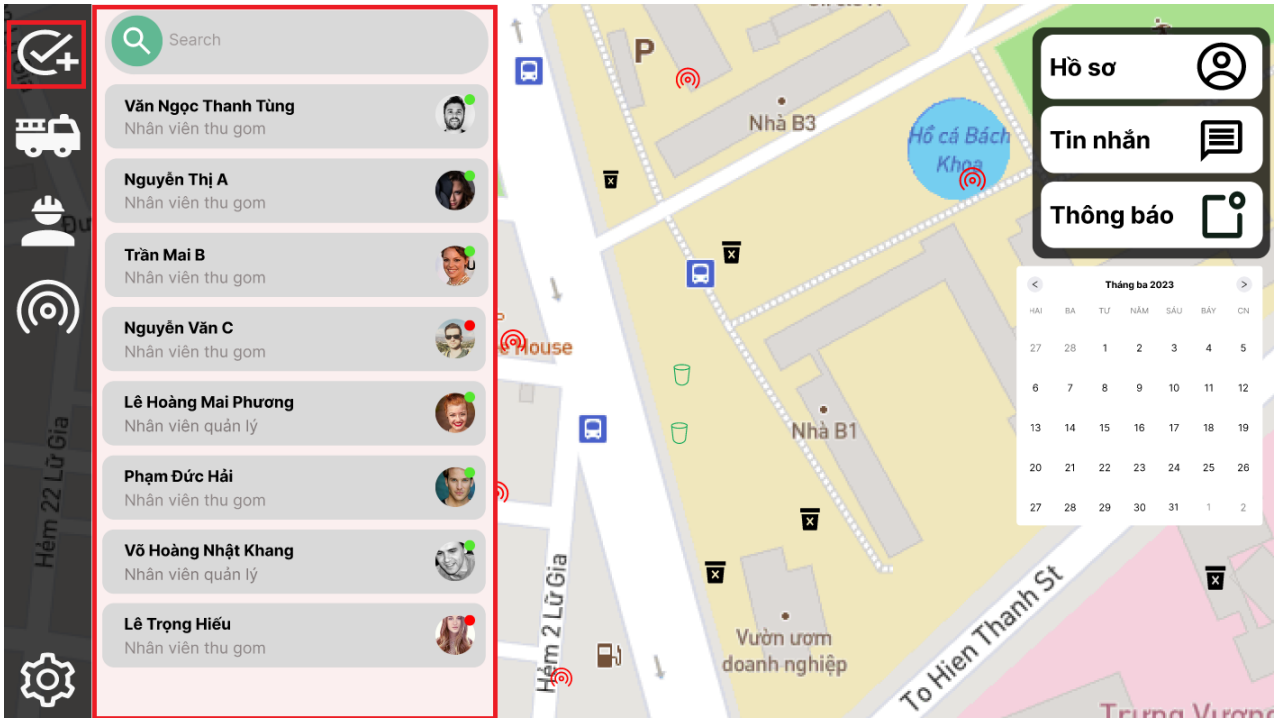



**Figure 2.8:** Back Officers starts assigning task to their workers

Here back officers can view their current status of workers, pick a worker and starts assigning tasks to them. Now, the back officer will click on the left button...


**Figure 2.9:** Back Officers view current workload of a worker

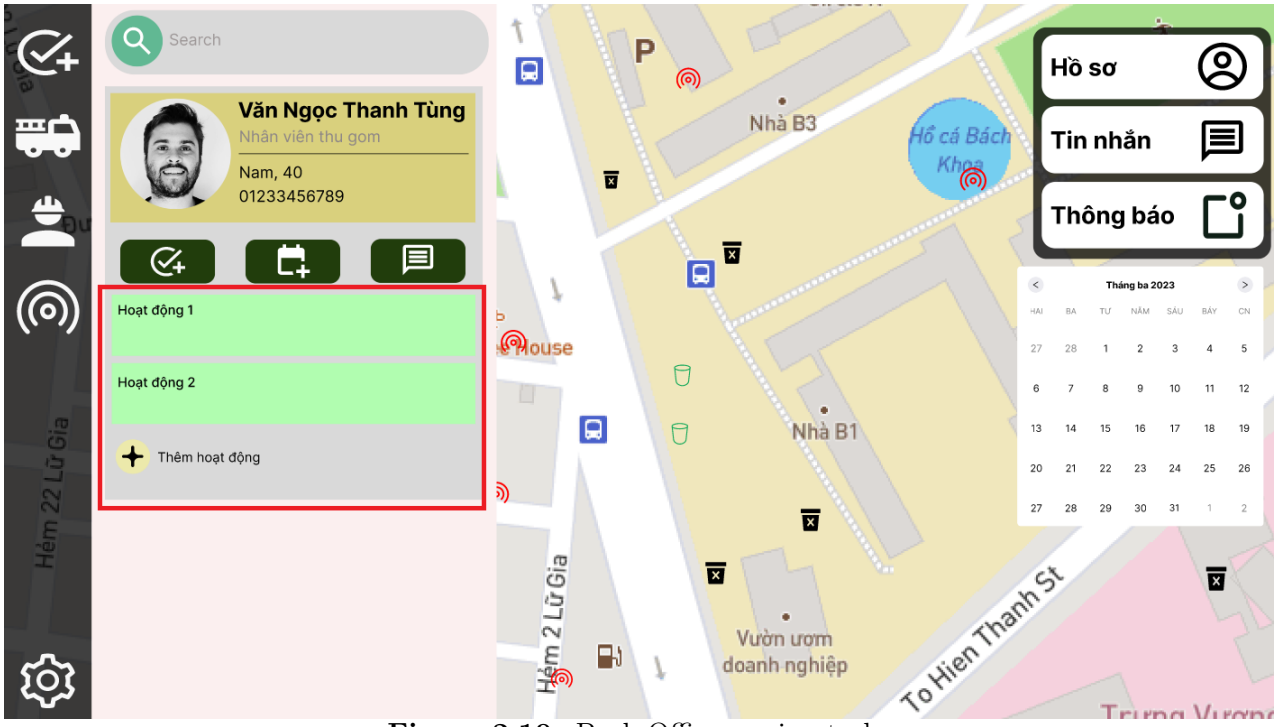



**Figure 2.10:** Back Officer assign tasks

### 2.4.4 Calendar Overview

Back officer can view schedule of workers by clicking on the "View schedule" button.


**Figure 2.11:** Back Officer views worker's working schedule

### 2.4.5 Message Overview

Back Officer can communicate with the workers through the message feature embedded in the current profile of the worker.

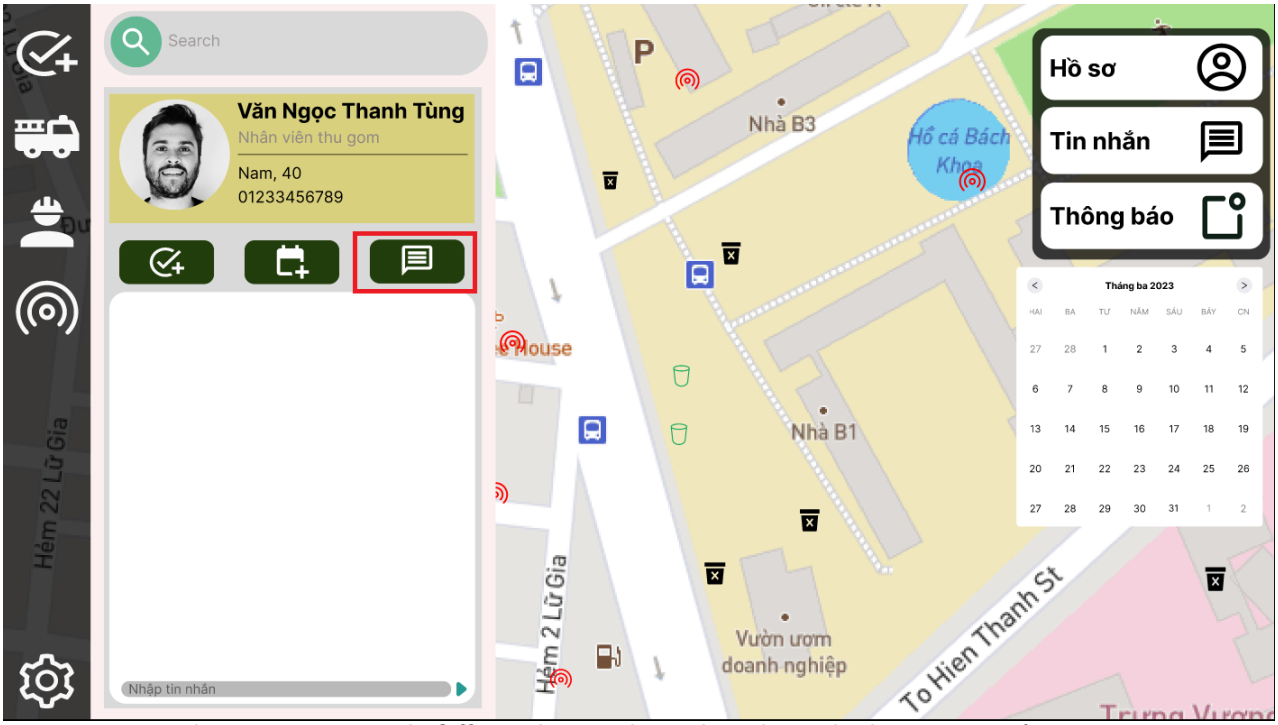



**Figure 2.12:** Back Officer chat with worker through the message feature

# Chapter 3

# Task 3: Architecture design

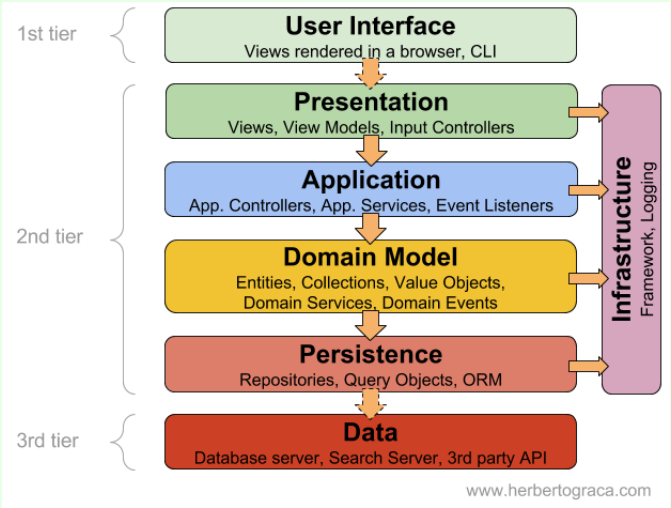
**3.1 Use a layered architecture to design the UWC 2.0 system. Describe how you will present your User Interface. Describe how you will store your data. Describe how you will access to external services/ APIs.**

### 3.1.1 Use a layered architecture to design the UWC 2.0 system

**Layered Architecture**

Layered architectures are said to be the most common and widely used architectural framework in software development. It is also known as an $n$-tier architecture and describes an architectural pattern composed of several separate horizontal layers that function together as a single unit of software. Components that are related or that are similar are usually placed on the same layers. However, each layer is different and contributes to a different part of the overall system.
The number of layers in a layered architecture is not set to a specific number and is usually dependent on the developer or software architect. It is important to note that this framework will usually always have a user interaction layer, a layer for processing, and a layer that deals with data processing.

- **Presentation Layer** – responsible for user interactions with the software system
- **Application/Business Layer** – handles aspects related to accomplishing functional requirements
- **Domain Layer** – responsible for algorithms, and programming components
- **Infrastructure/Persistence/Database Layer** – responsible for handling data, databases




UWC 2.0 system will consist of 5 layers. Each layers will contain some modules which will help functioning

the features of the web. Further description on the above architecture can be emphasized as follow:

1. **Database layer:** Store data, and a basic class to establish connections between the database and the persistence layer. Therefore, it has no module.
2. **Persistent layer:** Containing **Object Relational Mapping** (ORM) module, which is also a technique used in creating a "bridge" between object-oriented programs and, in most cases, relational databases. Put another way, we can see the ORM as the layer that connects Object-Oriented Programming (OOP) to relational databases.
3. **Service layer:** This layer will in charge of *externalizing* the from the **Business layer** to the corresponding services
4. **Business layer:** Containing 5 modules:
   - **Message module:** Manage the message features between the people working in the system.
   - **Task module:** This module manages the task assignment and view tasks.
   - **User module:** Manage users' authentication, profile,...
   - **MCP module:** Manage the MCPs information and reports
   - **Vehicle module:** In charge of vehicles current status, assigned workers,...
5. **User Interface layer:** Used for interacting process between user and the system, containing a view module

### 3.1.2 Describe how will you present your User Interface.

The way to present User Interface is quite intuitive. We may present the features of the web by the sidebar and the buttons.

To make it easier, we will provide some images below:

### 3.1.3 Describe how you will store your data

To store the data, we go for the MongoDB database. MongoDB Atlas is a fully-managed cloud database that handles all the complexity of deploying, managing, and healing your deployments on the cloud service provider of your choice (AWS , Azure, and GCP). MongoDB Atlas is the best way to deploy, run, and scale MongoDB in the cloud. With Atlas, you'll have a MongoDB database running with just a few clicks, and in just a few minutes.

In general, we tend to organize 6 main components in our database, including:

- `collector` and `janitor`: The below code is the `json` code, containing the attributes of objects `collector` and `janitor`. They contains general information such as `user_id` for indexing users, `first_name` and `last_name` for keeping user's name, `date_of_birth` for the date of birth of user, `gender`, `ava_url` is for avatar link, `phone_number`, `email`... Especially, the `task` attribute is an array to store a list of tasks of the worker. The `password` is for verifying user.

```
{
    "_id":
    {
        "$oid":"641c952998db414d7d1f975b"
    },
    "user_id": "",
    "first_name": "",
    "last_name": "",
    "date_of_birth": "",
    "gender": "",
    "tasks": [""],
    "ava-url": "",
    "email": "",
    "phone_number": "",
    "status": "",
    "password": ""
}

```

- `back-officer`: Except for the `tasks` array, the remaining attributes are the same.

```
        {
            "_id":
            {
                "$oid":"641d253e2490bc6b13a7847b"
            },
            "user_id": "",
            "first_name": "",
            "last_name": "",
            "date_of_birth": "",
            "gender": "",
            "ava-url": "",
            "email": "",
            "phone_number": "",
            "password": ""
        }

```

- `messages`: This collection is used for extracting users corresponding to their IDs, containing `from_userid` and `to_userid` of type string (used for matching users together), and `timestamp` is used for displaying the time the message sent.

```
        {
            "_id":
            {
                "$oid":"641d72462490bc6b13a7847e"
            },
            "timestamp": "",
            "from_userid": "",
            "to_userid": "",
            "content": ""
        }

```

- `tasks`: Describing `task` objects, containing an array of `workers` in charge of that task, the `priority` of the task (represented by numbers from 1 to 3, with 1 is the most emergency task needs to be resolved, and 3 is the least)

```
        {
            "_id":
            {
                "$oid": "641d7ef02490bc6b13a78483"
            },
            "workers": [""],
            "priority": "",
            "location": "",
            "latitude ": "",
            "longitude": ""
        }

```

- `vehicles`: Containing `vehicle_id`, array of `workers` on a single vehicle, an array of `MCPs`, and the status of that vehicle (used for checking the availability of vehicle).

```
        {
            "_id":
            {
                "$oid": "641d870e2490bc6b13a78486"
```

```
            },
            "vehicle_id": "",
            "workers": [""],
            "to_MCP": [""],
            "status": ""
        }

```

### 3.1.4 Describe how you will access to external services/ APIs

In this project, the external API will be the map. It will be contained in the service module, while having other features on it. The map can be shown on the dashboard. For example, on the map, we can view some information such as MCPs, vehicles, workers,...

Our very first step is getting started with **Google Cloud Platform** in order to access to the Google Map product. We choose Google Cloud because this is a very useful platform for its variety of features, and of course, having Google Map API which is useful for implementing map feature in `reactjs` (https://www.npmjs.com/package/@react-google-maps/api).

Next is to get Map API (https://developers.google.com/maps/documentation/javascript/adding-a-google-map)

The code for managing our Google Map API is as follow:

```
<main className="BO--content">
    {isLoaded && (
      <div
        className="map"
        // onClick={() =>
        //   showSidebar
        //     ? setShowSidebar(!showSidebar)
        //     : setShowSidebar(showSidebar)
        // }
      >
        <GoogleMap
          mapContainerStyle={containerStyle}
          center={center}
          zoom={zoom}
          options={mapOptions}
        >
          <Marker position={center} />
          <></>
        </GoogleMap>
      </div>
    )}
</main>
```

**Listing 3.1:** Google Map API management

## 3.2 Draw a component diagram for the Task Assignment module

**Component diagram**

**Component diagrams** are used in modeling the physical aspects of object-oriented systems that are used for visualizing, specifying, and documenting component-based systems and also for constructing executable systems through forward and reverse engineering. Component diagrams are essentially class diagrams that focus on a system's components that often used to model the static implementation view of a system.
A component represents a modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. In UML 2, a component is drawn as a rectangle with optional compartments stacked vertically. A high-level, abstracted view of a component in UML 2 can be modeled as:

1. A rectangle with the component's name
2. A rectangle with the component icon
3. A rectangle with the stereotype text and/or icon

Two type of component interfaces include:

- **Provided interface** symbols with a complete circle at their end represent an interface that the component provides - this "lollipop" symbol is shorthand for a realization relationship of an interface classifier.
- **Required interface** symbols with only a half circle at their end (a.k.a. sockets) represent an interface that the component requires (in both cases, the interface's name is placed near the interface symbol itself).

**The subsystem classifier** is a specialized version of a component classifier. Because of this, the subsystem notation element inherits all the same rules as the component notation element. The only difference is that a subsystem notation element has the keyword of subsystem instead of component.
**Ports** are represented using a square along the edge of the system or a component. A port is often used to help expose required and provided interfaces of a component.
Graphically, a component diagram is a collection of vertices and arcs and commonly contain components, interfaces and dependency, aggregation, constraint, generalization, association, and realization **relationships**. It may also contain notes and constraints.

### 3.2.1 Diagram

Follow this link to view more: https://drive.google.com/file/d/13WUMfb9aMruG2aWSKsgsfU_pUex91HGP/view?usp=sharing

### 3.2.2 Description

The diagram can be separated into 4 different layers:

#### 3.2.2.1 Presentation layer:

- `Log in View:` Render the `Account authentication` interface from component `Account Logic` (internal component `Authentication`)
- `Message View:` Render the `Message` interface exposed by component `Message Logic`.
- `Calendar View:` Render the calendar interface to user, in which will contain the informations of daily/weekly tasks of user. Exposed from the `Get Task` logic.
- `Back Officer View:`
  - `Overview View:` Render general information for back officer by using `Employees' information` provided by component `Account Logic` (internal component `Account Management Logic`) and `Tasks' overview` information provided by component `Task Logic` (internal component `Task management Logic`).

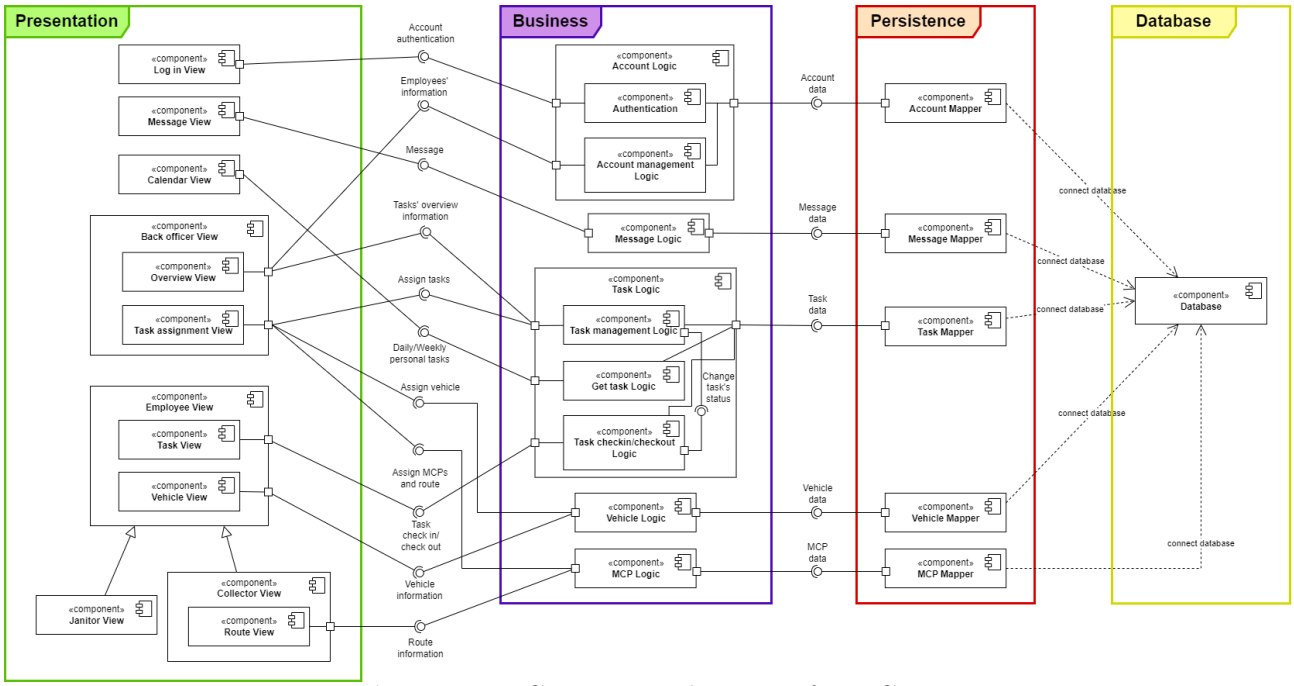



**Figure 3.1:** Component diagram of UWC 2.0

  - `Task assignment View`: Render the task assignment view for back officer, requires `Assign tasks`, `Assign vehicle` and `Assign MCPs and route` from components `Task Logic` (internal component `Task management Logic`), `Vehicle Logic` and `MCP Logic`, respectively.

- `Employee View`:

  - `Task View`: Displays individual task and its details to collector and janitor by using `Task information` and `Task check in / check out` from component `Task Logic` (internal component `Get task Logic` and `Task check in/check out Logic`, respectively).
  - `Vehicle View`: Render vehicle information to collector and janitor by using `Vehicle information` from component `Vehicle Logic`.
  - `Component Employee View`: Inherited by component `Janitor View` displaying UI for janitor, and component `Collector View` displaying UI for collector. Component `Collector View` has another internal component `Route View` displaying the route UI for collector by using `Route information` from component `MCP Logic`.

#### 3.2.2.2 Business layer:

Consists of 5 major components representing 5 modules of the Business layer we have discussed earlier:

- `Account Logic`: Handle logics of the authentication aspect
- `Message Logic`: View and store message
- `Task Logic`: Handles task view, task assignments,...
- `Vehicle Logic`: Handles vehicle assignment logics
- `MCP Logic`: In charge of watching MCP situation, reports,..

Each of which requires the data object from its respective data mapper component to perform its business logic.

#### 3.2.2.3 Persistence layer:

Consists of 5 data mappers:

- `Account Mapper`: Map any service requests from `Account Logic` to the database
- `Message Mapper`: Map any service requests from `Message Logic` to the database
- `Task Mapper`: Map any service requests from `Task Logic` to the database
- `Vehicle Mapper`: Map any service requests from `Vehicle Logic` to the database
- `MCP Mapper`: Map any service requests from `MCP Logic` to the database

All of which establish a connection to `Database` in order to retrieve and perform query on the data.

#### 3.2.2.4 Database layer:

Consists of database module providing data to each data mapper in the `Persistence` layer

# Chapter 4

# Task 4: Implementation – Sprint 1

## 4.1 Setting up an online repository (github, bitbucket, etc) for version control.

**Check for our official repository here**: https://github.com/nhatkhangcs/UWC2.0-TheFinalists.git

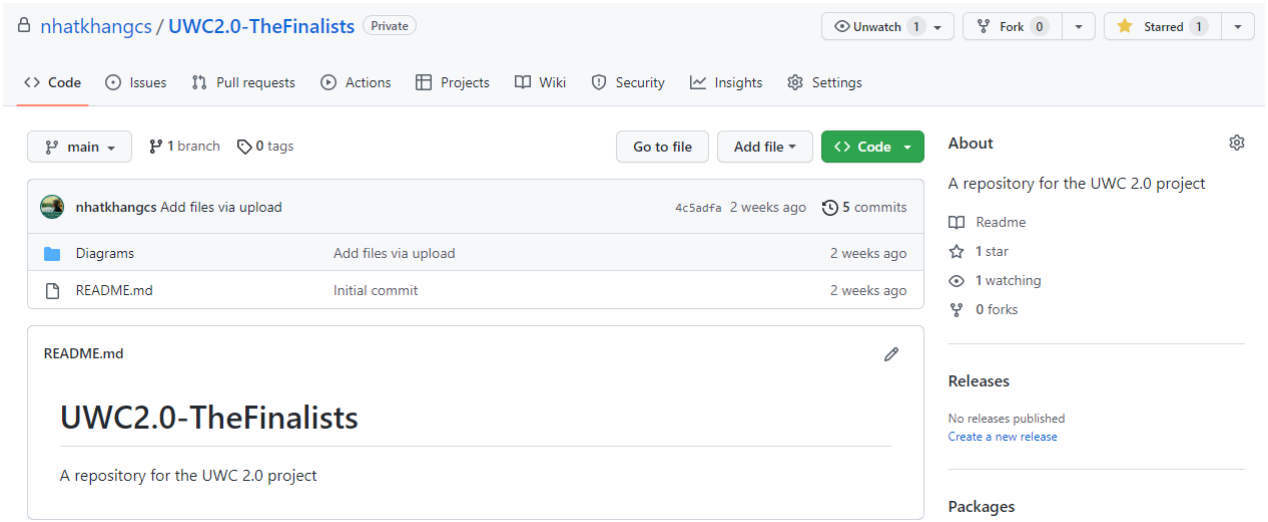



**Figure 4.1:** Initialize repository

## 4.2 Adding documents, materials and folders for Requirement, System modelling and Architectural design. Use the selected version control system to report the changes to these files.

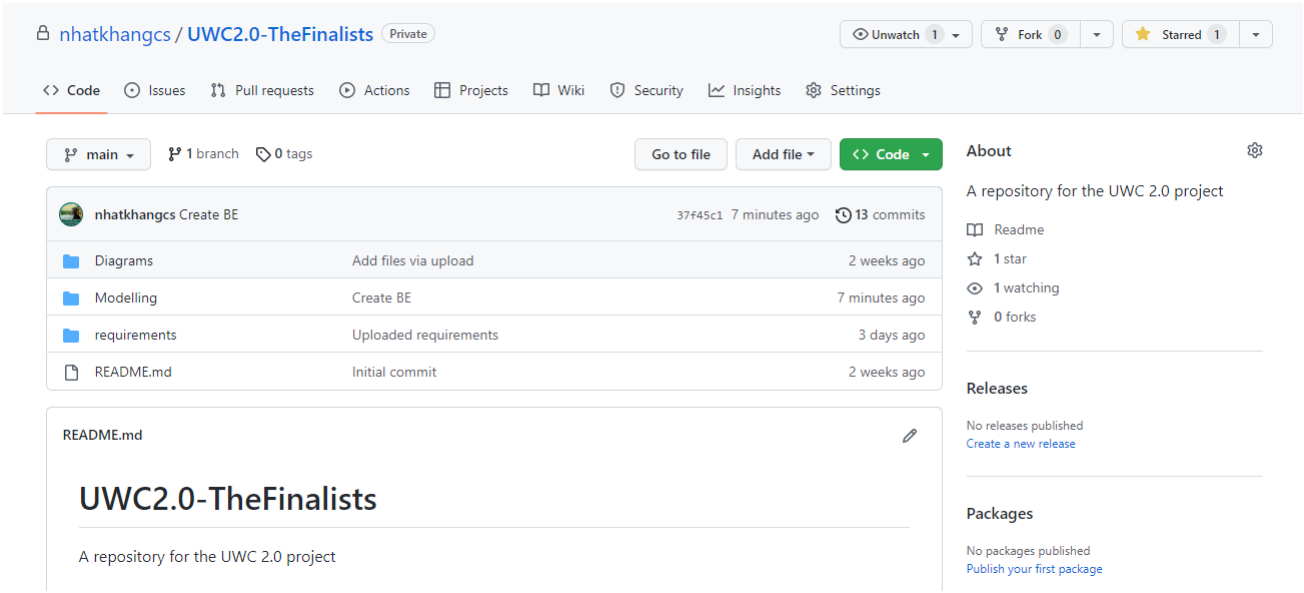



**Figure 4.2:** Adding documents into the repository

Here we decided to contain 3 folders in our repository:

1. **Diagrams:** Containing designs pattern for the UWC 2.0 system as the following figure:

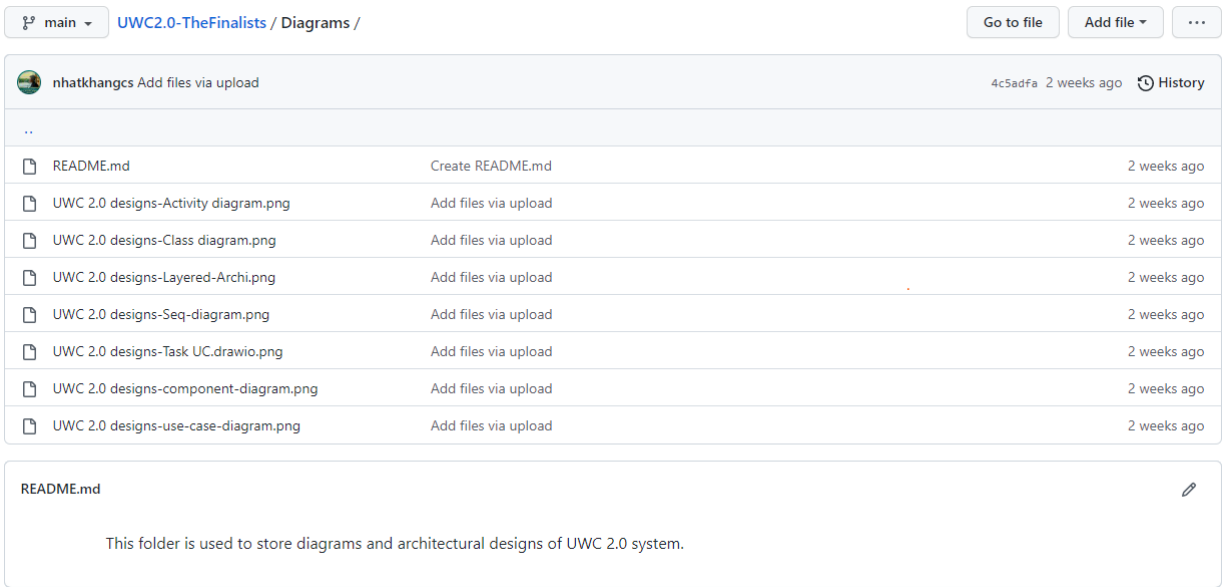



**Figure 4.3:** Diagrams folder

2. **Requirements:** Containing 2 files, one for general requirements and one for detail requirement

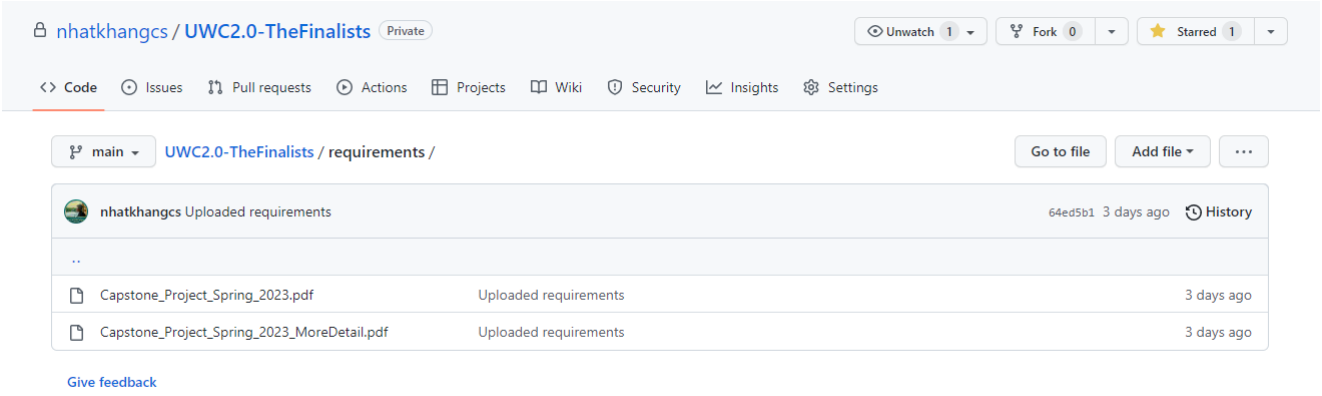



**Figure 4.4:** Requirements folder

3. **Source code for modelling:** Here we tends to divide into 2 folder, **FE** (stands for Front-End) and **BE** (stands for Back-End)

   3.1 **Front-End**: This directory contains a public folder which holds public images, a folder that contains source code, two sets of package requirements for developers to install once pulled.

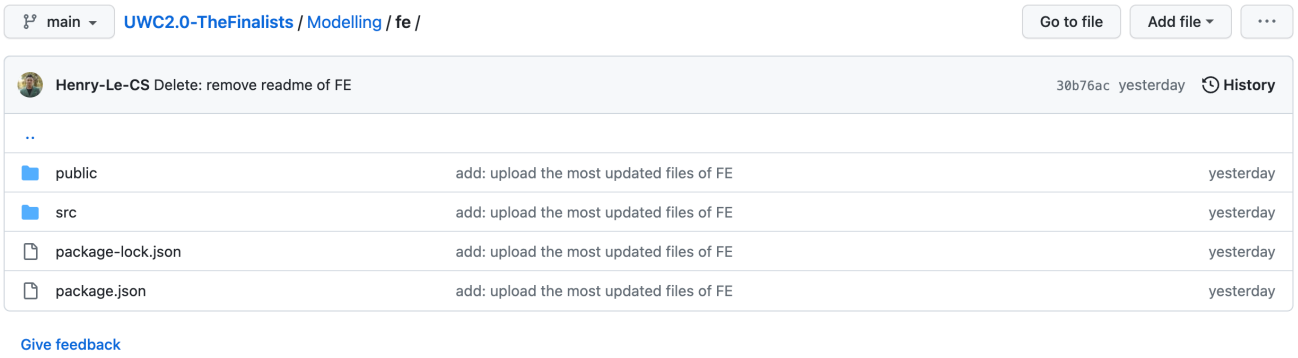



**Figure 4.5:** Front-End folder

Inside Source Code directory, we can see there are 4 other directories and 3 different files.

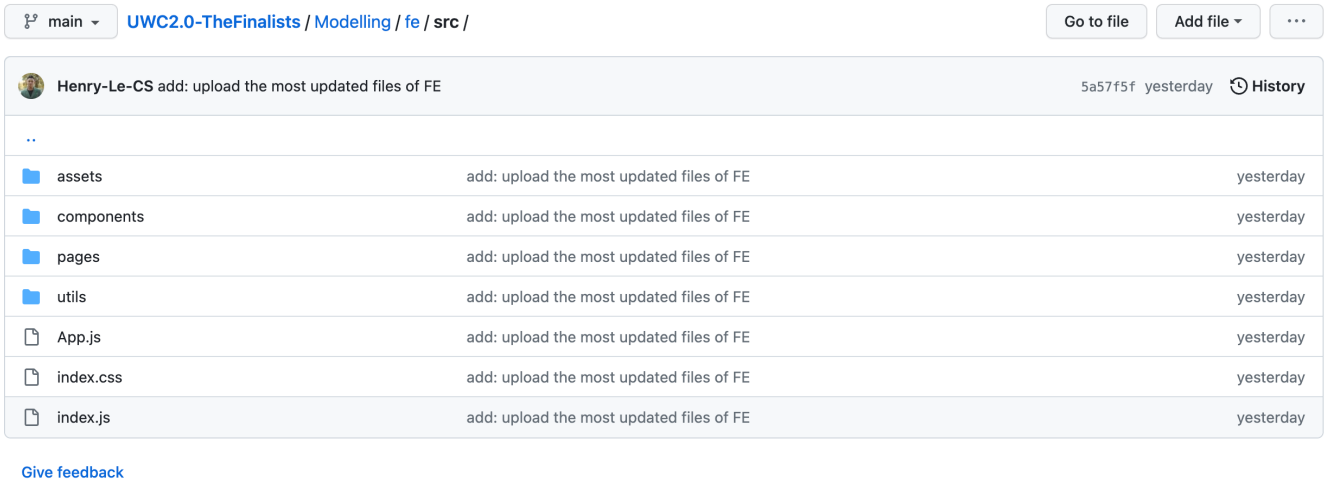



**Figure 4.6:** Source code folder

- **Assets**: This folder contains images, icons that we do not want to share while the website is rendered.

- **Pages**: This folder contains 5 sub-folders that contain five sites of our web applications, for each there are two JSX and CSS files.

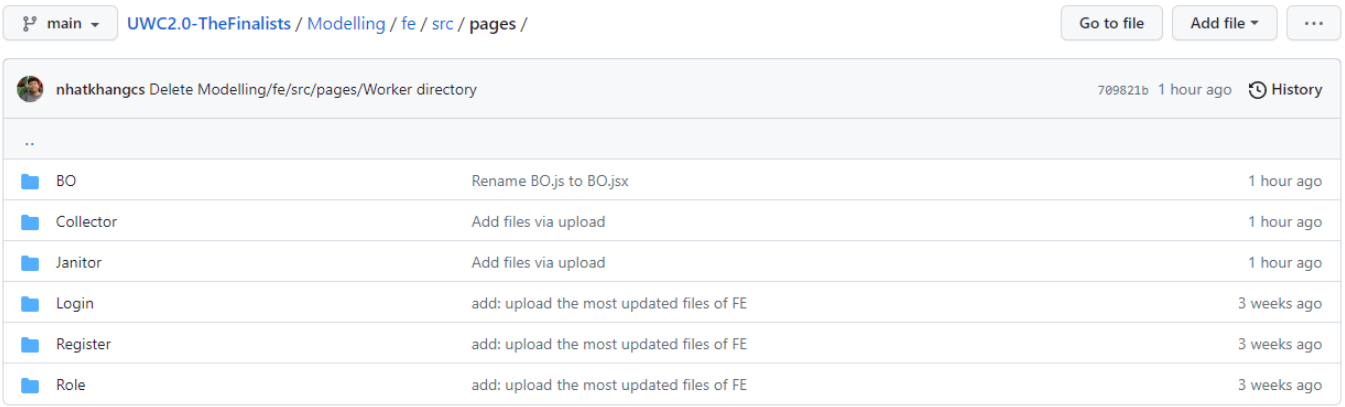



**Figure 4.7:** Pages folder

Particularly, the Role, Login, Register is used for selecting roles and login/register, BO and Worker are dashboards for Back Officers, Janitors and Collectors

- **Components**: This folder contains several components for mutual use between Back officers and Workers. Inside each components there may also be Child components to support code splitting.

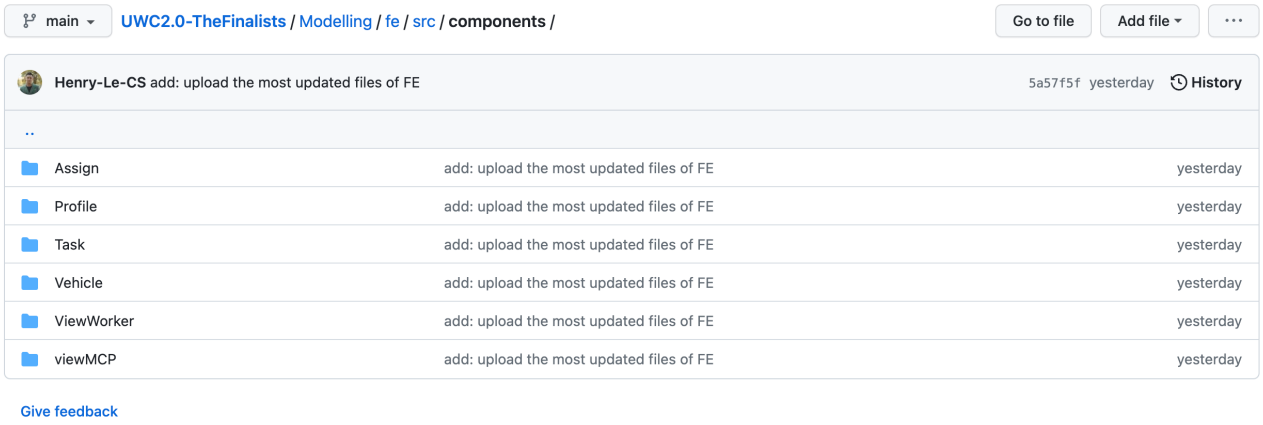



**Figure 4.8:** Component folder

- **Utils**: This directory contains functions that might be shared for Back Officers and Workers
- **App.js**: This file is where all the Pages are assembled and prepared for rendering.
- **Index.js** and **Index.css**: Those files are the root where App.js's components are rendered. The CSS files are included for some general edition of the UI.

3.2 **Back-End**: Similar, but not identical structures can also be seen in the Back-End directory. This directory contains 4 sub-directories and 3 important files for the Server side to function well.

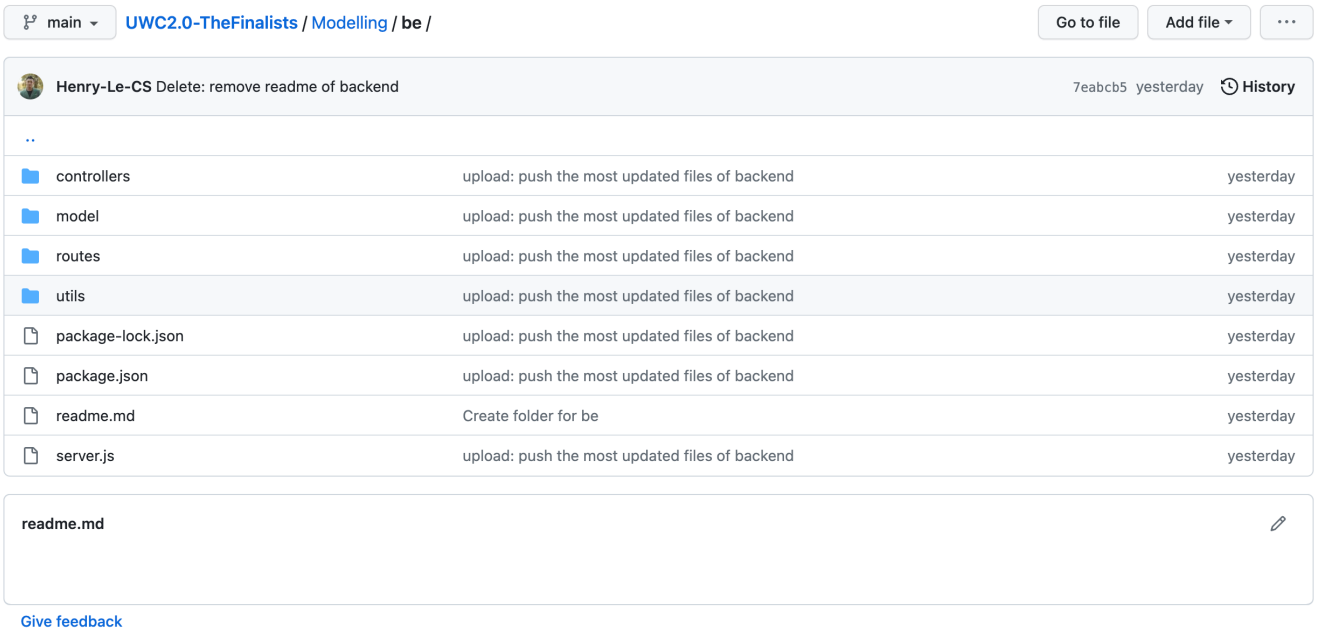



**Figure 4.9:** Back-End folder

- **server.js**: This file is the soul of Server-side as it hosts the app and re-direct the currect routes to fit the URL
- **routes**: This folder contains three possible routes regarding User, Back Officer and Worker and then map them to the appropriate Controllers to perform an action.

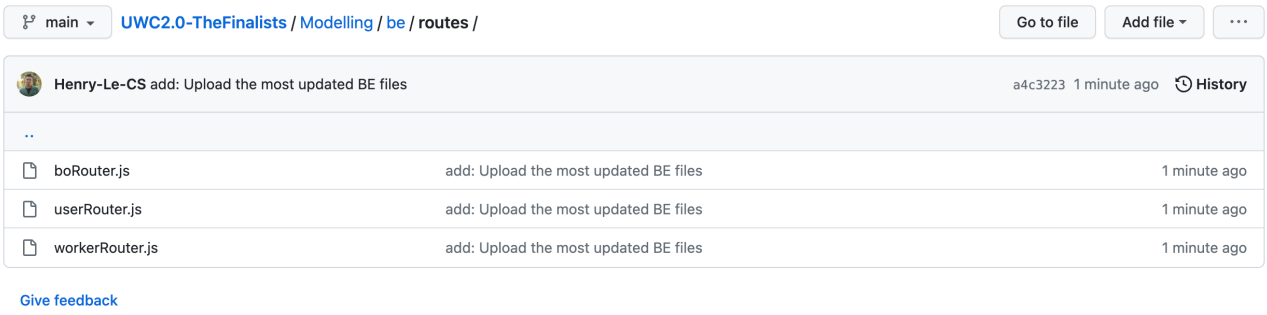



**Figure 4.10:** Routes folder

- **controllers**: This folder contains three Controllers regarding User, Back Officer and Worker. For each directory and parameters passed to the URL, it will direct to a particular controller functions to execute.

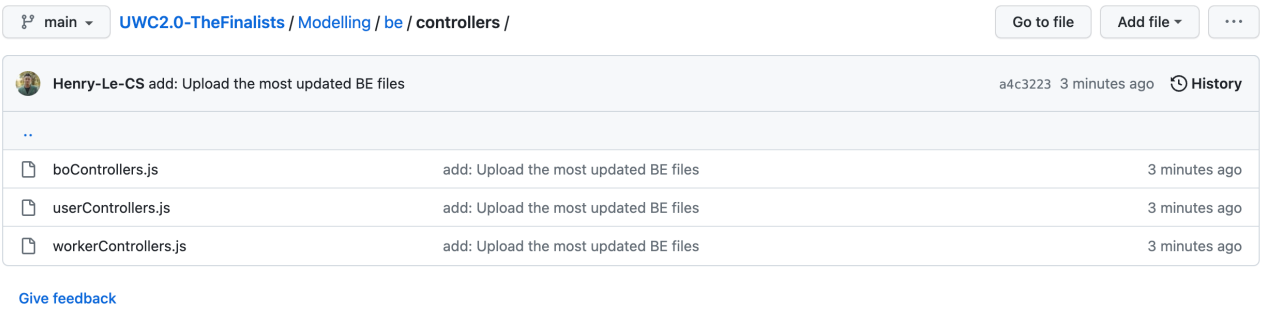



**Figure 4.11:** Routes folder

- **models**: This folder contains a single data.js files that return a pointer that connects to database. It is called when Controller functions want to manipulate with the data in database

- **utils**: This folder contains functions that are used mutually and repeatedly.
- **package-lock.json, package.json**: These two files contains the necessary dependencies for developers to install once they pull.

**4.3 Conduct an usability test with the user interface you developed in MVP1. Summarize the feedback and improve the MVP1 into MVP2 (with better User Experience).**

### 4.3.1 Conduct an usability test with the user interface you developed in MVP1

To conduct this test, we decided to create a platform and took survey of people using our webpage to see if they like our webpage or not. Most of users feel comfortable with the UIs of MVP1, just a touch needs of improvement to have a better look at MVP2. Figma is quite hard to demonstrate to the users what we intended to do with the webpage. The questions in the google form is used mainly for gathering most critical ideas and how **naive** (or, easy) the UIs can become in their point of view. After this survey, changes have been made to make the system more UI-friendly and serve better user-experience.

1. **The Google Form can be viewed here:** https://forms.gle/h3RFU8yeS1NEiTdM8
2. **Tester report:** https://www.overleaf.com/read/vzsrppxjsyst

#### 4.3.1.1 Login/Register

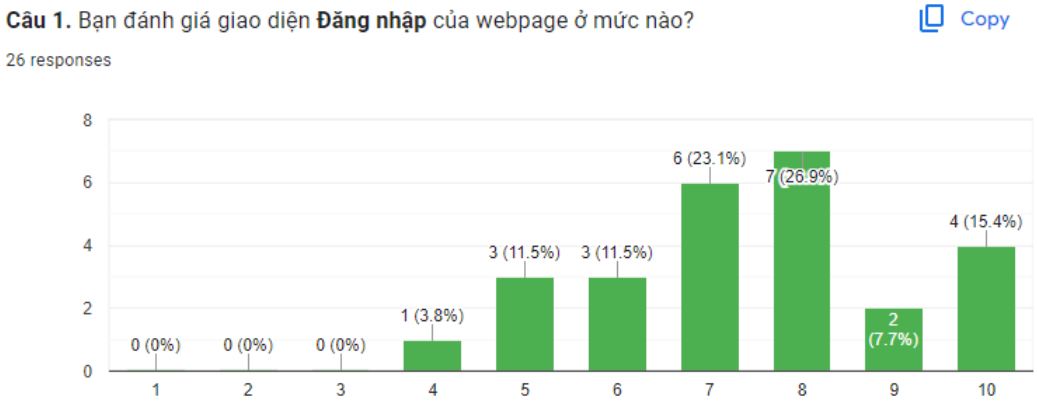



**Figure 4.12:** Login UI survey

#### 4.3.1.2 Dash Board

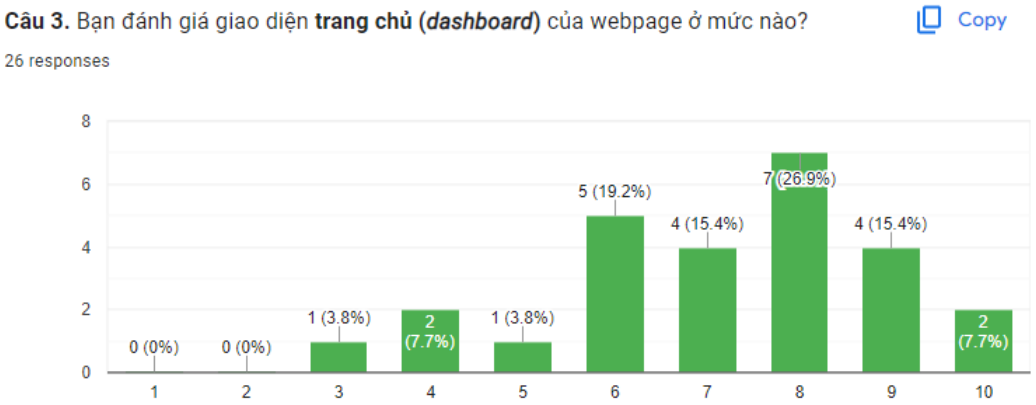



**Figure 4.13:** Personal Information UI survey

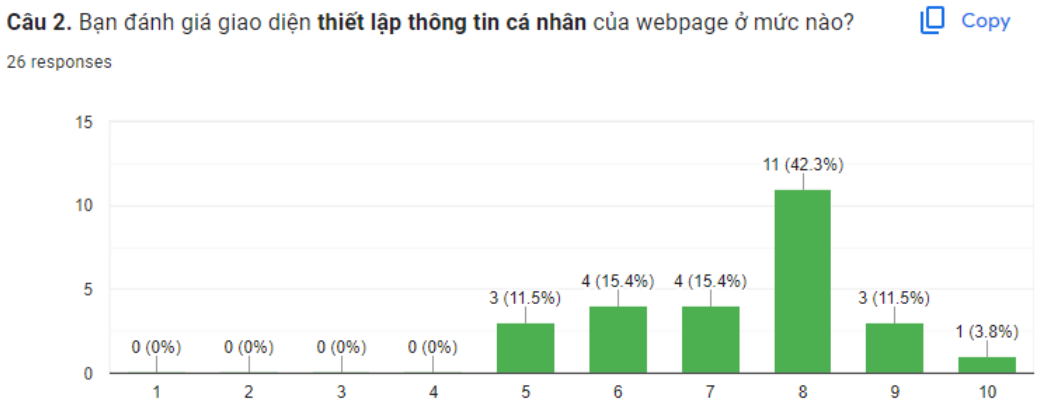



**Figure 4.14:** Personal Information UI survey

#### 4.3.1.3 Set personal information UI

#### 4.3.1.4 Assign tasks

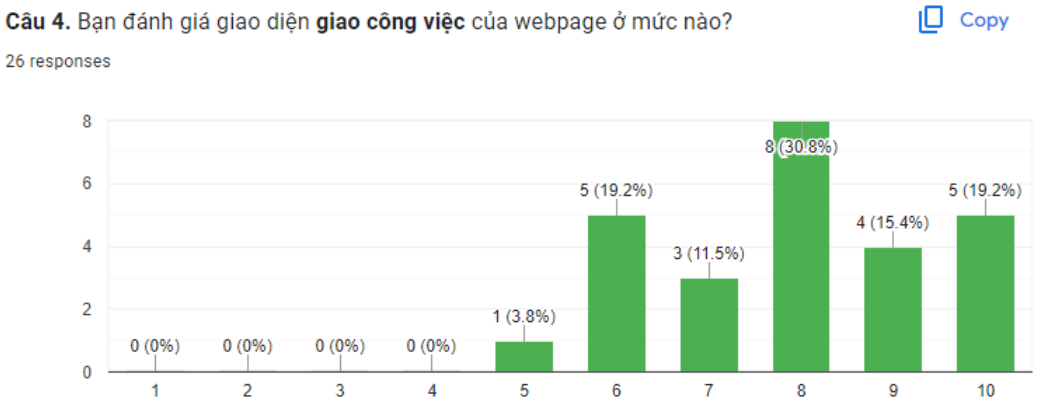



**Figure 4.15:** Personal Information UI survey

#### 4.3.1.5 Assign vehicles

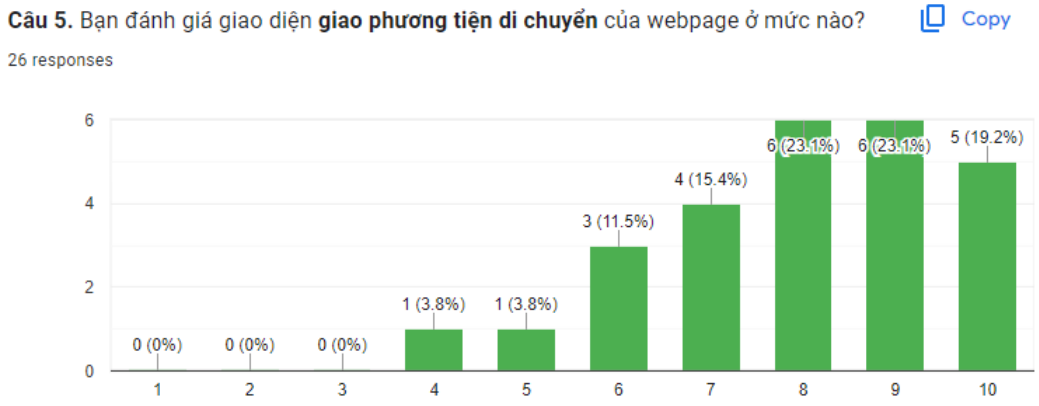



**Figure 4.16:** Assign vehicle UI survey

#### 4.3.1.6 View profiles

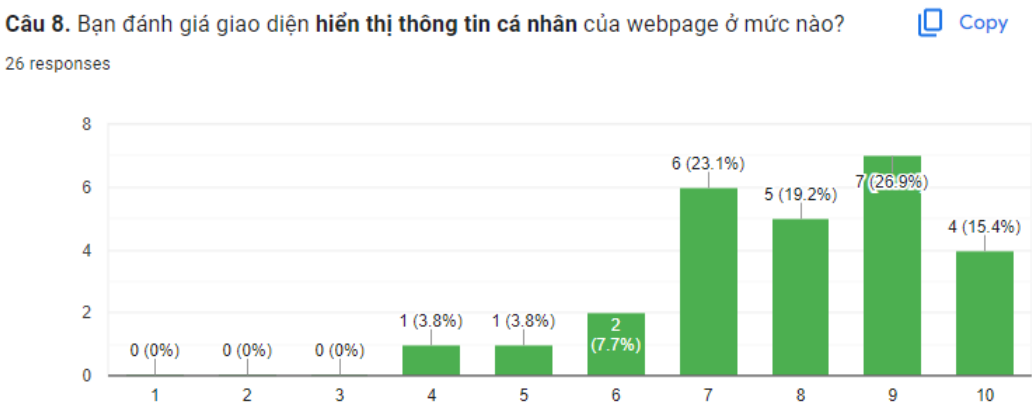



**Figure 4.17:** View profile UI survey

#### 4.3.1.7 View workers (from Back officer view)

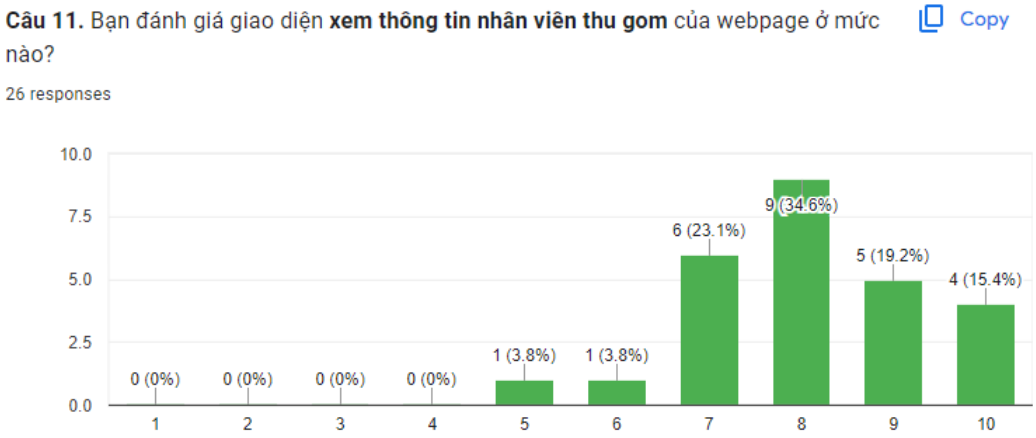



**Figure 4.18:** View worker information UI survey

#### 4.3.1.8 Map

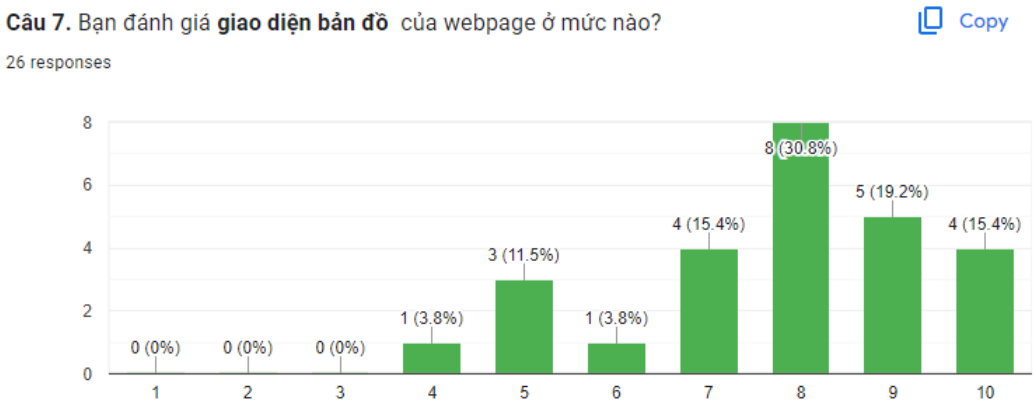



**Figure 4.19:** View map UI survey

#### 4.3.1.9 View MCPs

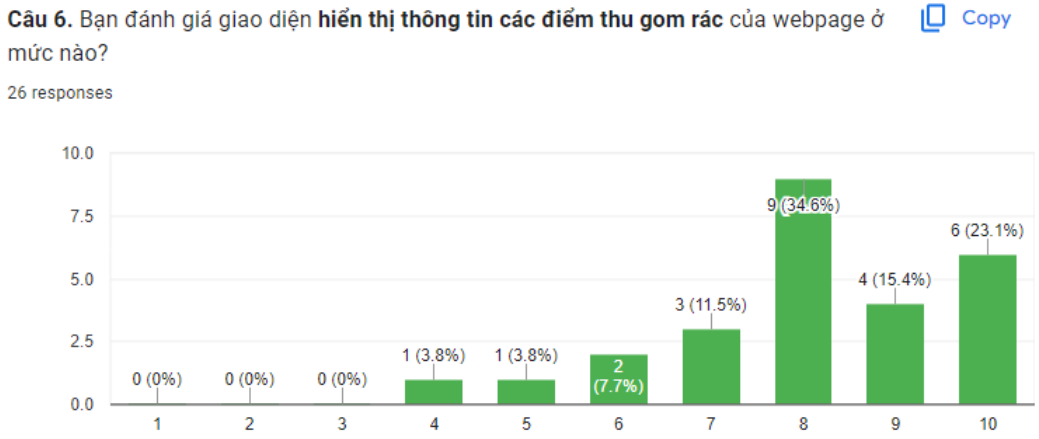



**Figure 4.20:** View MCPs information UI survey

#### 4.3.1.10 Chat

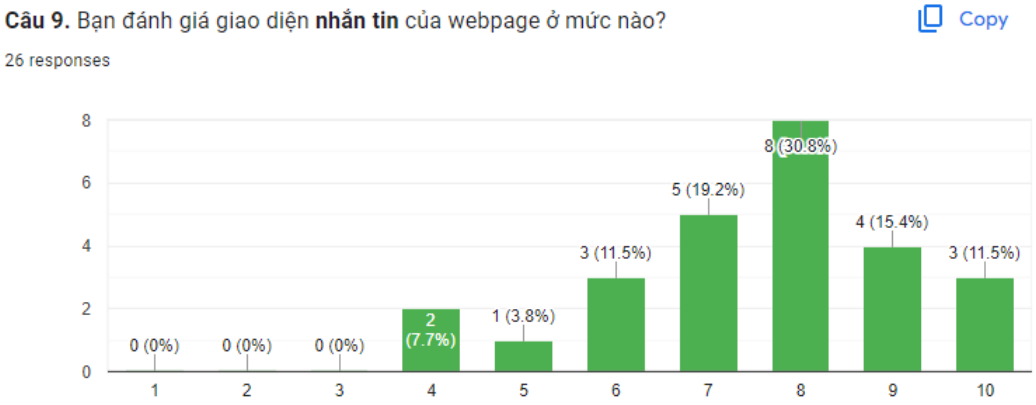



**Figure 4.21:** View chat UI survey

#### 4.3.1.11 View notification

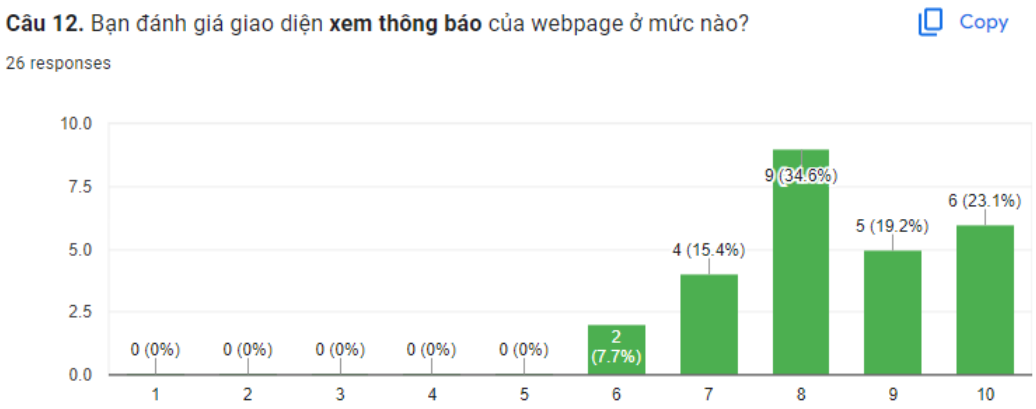



**Figure 4.22:** View notification UI survey

### 4.3.2 Summarize the feedback and improve the MVP1 into MVP2 (with better User Experience)

Overall, the survey result is quite attractive, average score of users is 8 out of 10.

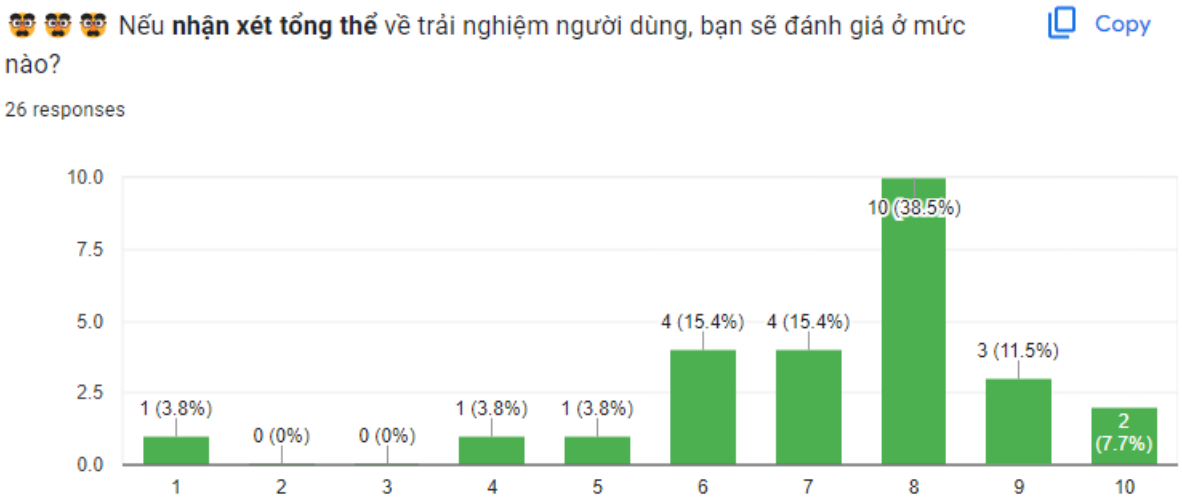



**Figure 4.23:** Overall score of UI-UX

And then, about whether users can themselves assign tasks and vehicles to the workers, most of them confirm that they can.

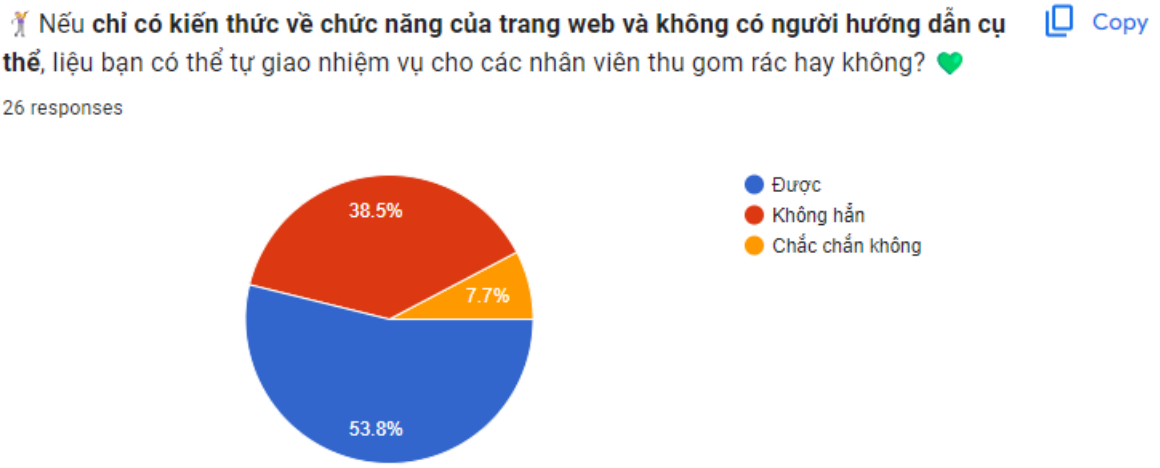



**Figure 4.24:** How naive is our UI to the users?

After summarizing all the feedback, we have come up with some most critical ideas about our UIs:

1. **Improvements:**

   - **Dashboard**: It's better to create 2 button at both sides of the screen first. When the user clicks on the button, then the menu appears, left-side or right-side. So we decided to improve the current UI to something like the figure below. But there are some comments that the map should be implemented using Satellite technology. What's more important is that: Since we will handle 10.000 MCPs in the future, it is important that the map accounts for the biggest visual part of our webpage.

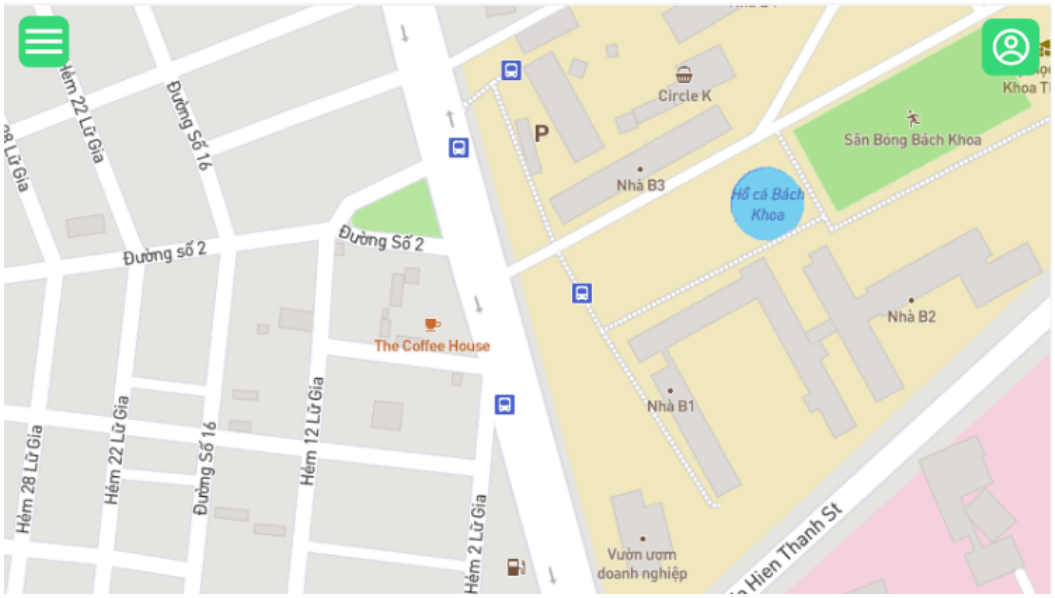



**Figure 4.25:** Improved UI for dashboard

- **Chat UI and View Profile UI:**

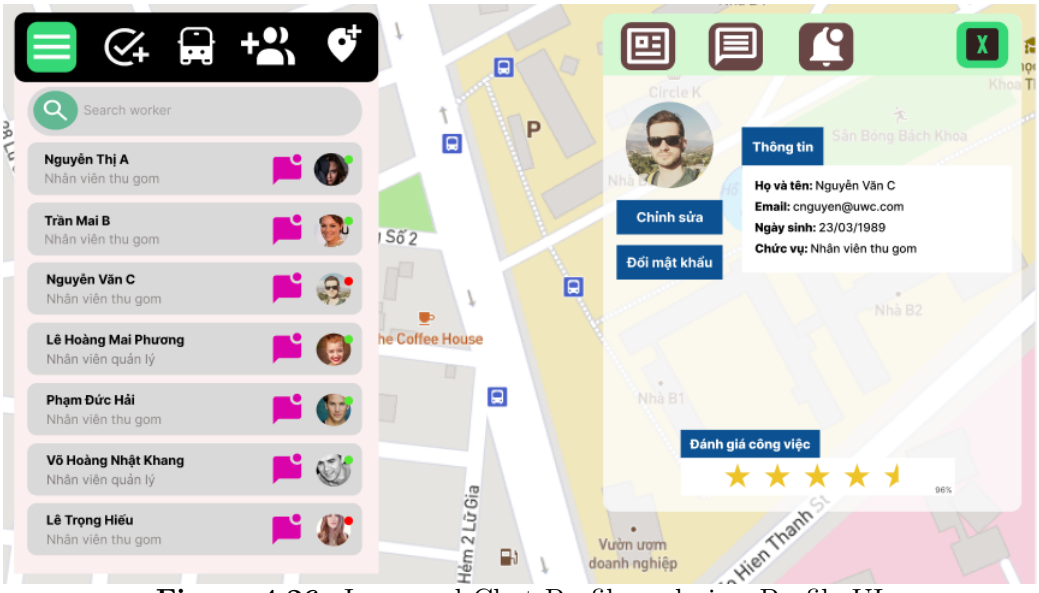



**Figure 4.26:** Improved Chat Profile and view Profile UI

- **View vehicles UI:**

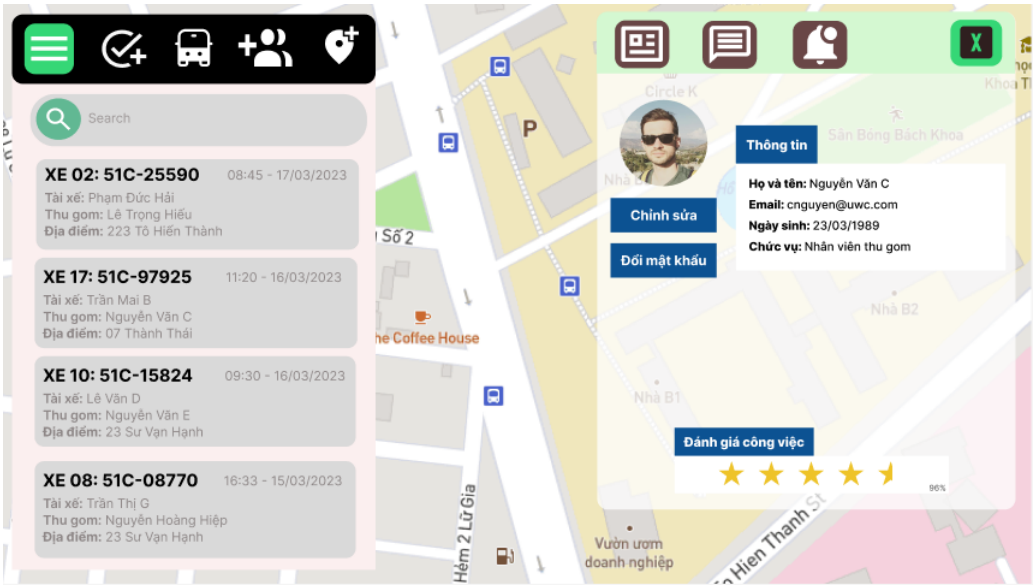



**Figure 4.27:** Assigning vehicles

- **View message and daily tasks:**

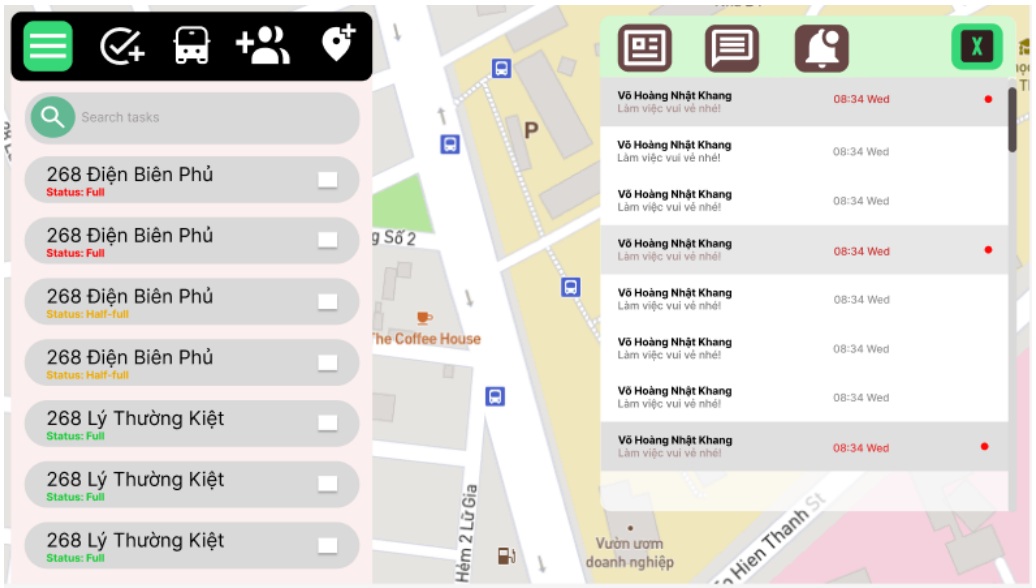



**Figure 4.28:** View message and tasks

- **View notifications and the MCPs status:**

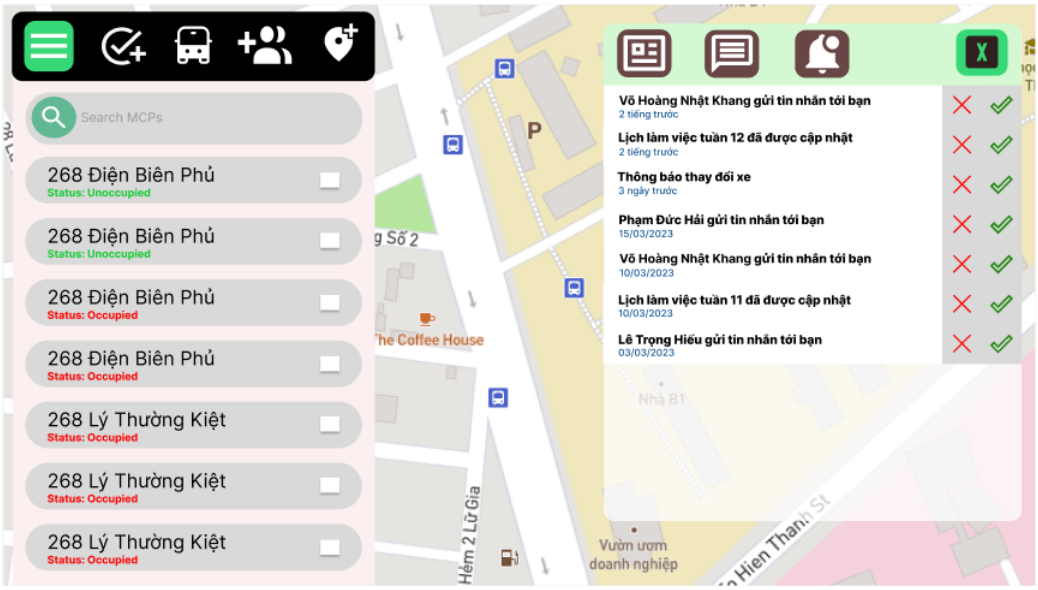



**Figure 4.29:** View notification and MCPs

2. **Advanced features (in development mode and quite hard to describe it on Figma):**

   - **View live MCP:** It is a little feature that helps Back Officers view MCP's current location and status easier. Just by clicking on the MCP name on the sidebar, the map will automatically redirect the center to the location
   - **Show assigned routes:** Show back officers and workers assigned route on the map
   - **View tasks through calendar:** Worker can view their tasks through calendar, and Back officer can view workers' tasks through their calendar too

# Chapter 5

# Task 5: Implementation – Sprint 2

**5.1 Develop MVP3 by implementing (with both frondend and backend) the interface in MVP2. You are free to choose the programming language (HTML, Javascript, Python, C#, etc). It is not required to implement a database in the backend. Data can be hard coded in code files.**

For our project, we have decided to use ReactJS for our web. ReactJS is a popular JavaScript library that is commonly used for building user interfaces on the web. Several reasons why ReactJS is a great choice for web development:

- **Component-based architecture:** ReactJS is designed around the concept of reusable, modular components. This makes it easy to build complex user interfaces that can be easily maintained and updated.
- **Virtual DOM:** ReactJS uses a virtual DOM (Document Object Model) to keep track of changes to the user interface. This allows it to update the UI efficiently and quickly, without having to reload the entire page.
- **Declarative programming:** With ReactJS, you can describe how the UI should look at any given time, and React takes care of updating the actual UI based on those descriptions. This makes it easier to reason about how the UI should behave and makes it less error-prone than imperative programming.
- **Large community and ecosystem:** ReactJS has a large and active community of developers who are constantly creating new tools, libraries, and resources. This means that it is easy to find help, learn new techniques and best practices, and integrate ReactJS into your development workflow.
- **Cross-platform compatibility:** ReactJS can be used to build web applications that work across different browsers and devices, as well as mobile applications for iOS and Android. This makes it a versatile and powerful tool for building applications that need to reach a wide audience.

For the backend, we use MongoDB, with reasons explained in Task 3. Besides, the following UIs will be implemented for better user experience:

### 5.1.1 Assigning task UI

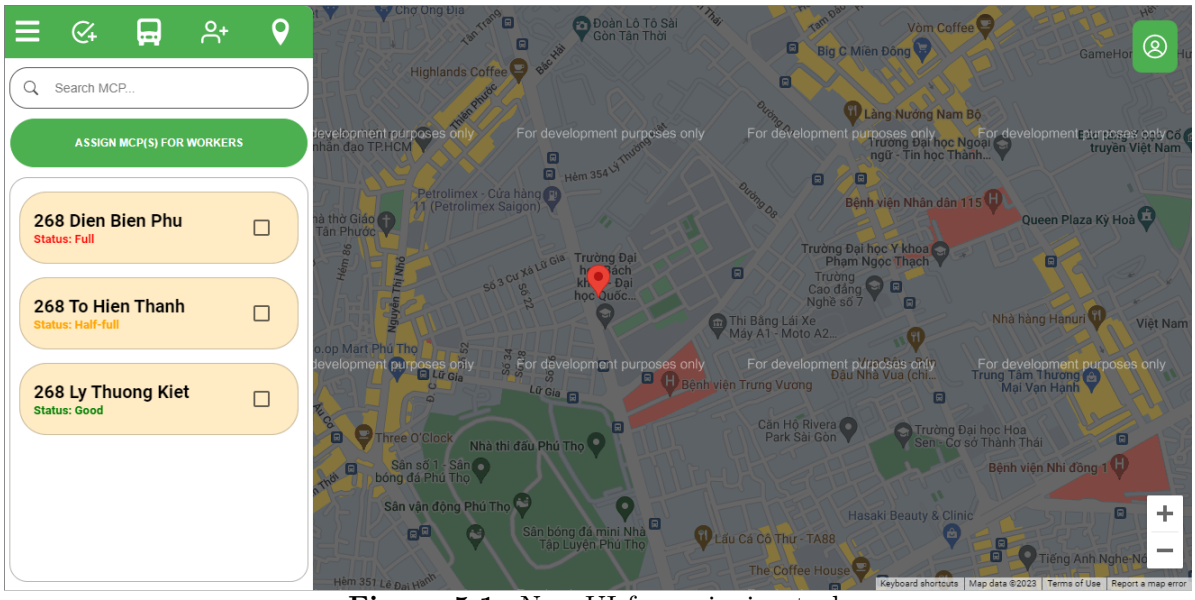



**Figure 5.1:** New UI for assigning task

With this new view, Back Officers will be able to view MCP address as well as their status quite conveniently. They will choose a MCP up to their choice, and click "Assign MCP(s) for workers". A box will appear on the screen:

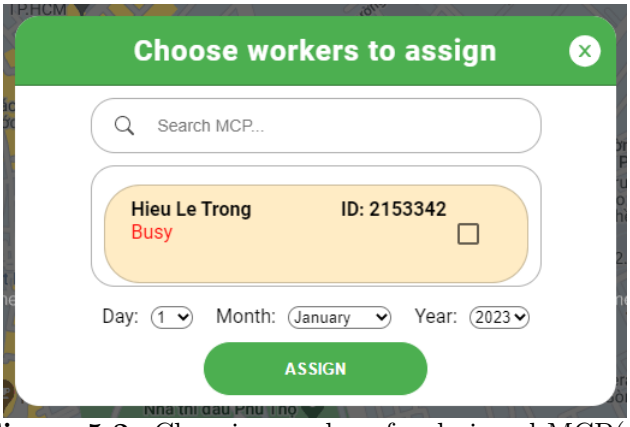



**Figure 5.2:** Choosing workers for designed MCP(s)

And this is where Back Officers will choose workers.

### 5.1.2 Assigning vehicle UI

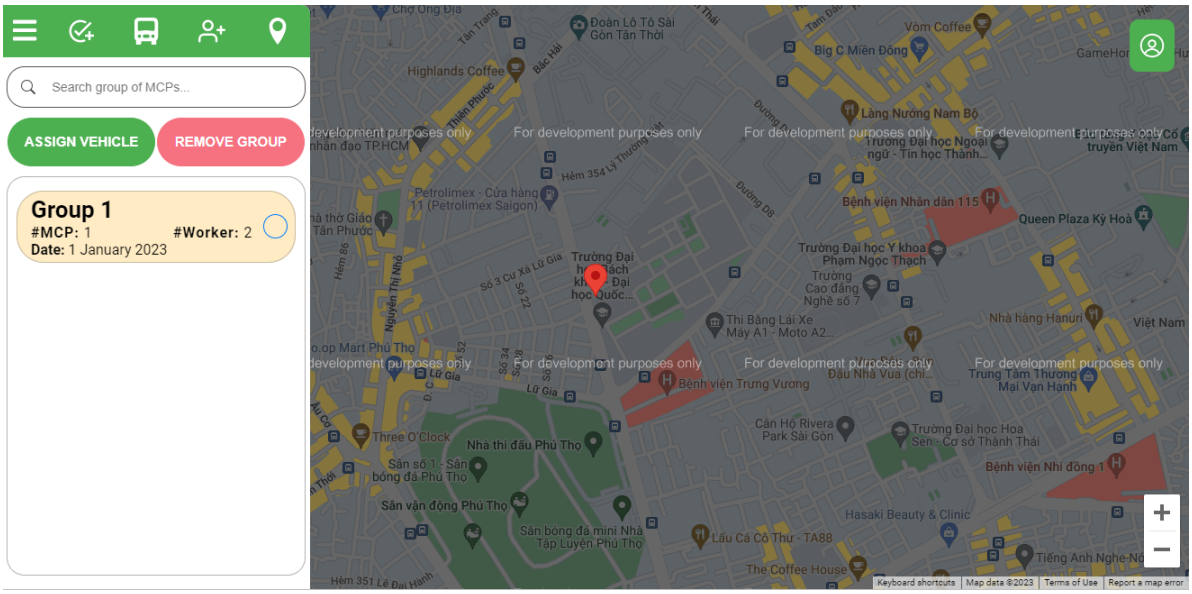



**Figure 5.3:** New UI for assigning vehicles

After choosing workers, Back Officers will now choose vehicle for them to drive to the MCP.

### 5.1.3 Worker details

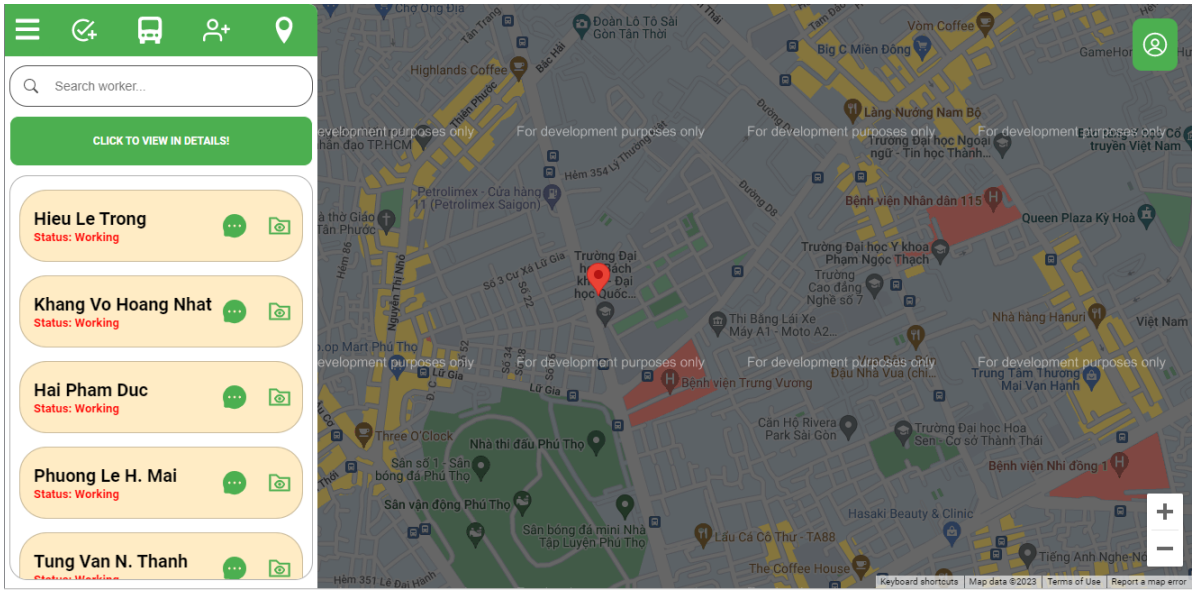



**Figure 5.4:** Worker tab on the left sidebar

By accessing to this tab, Back officers can start a conversation with worker by clicking the chat button, or they can view personal information by clicking the icon next to the chat icon:

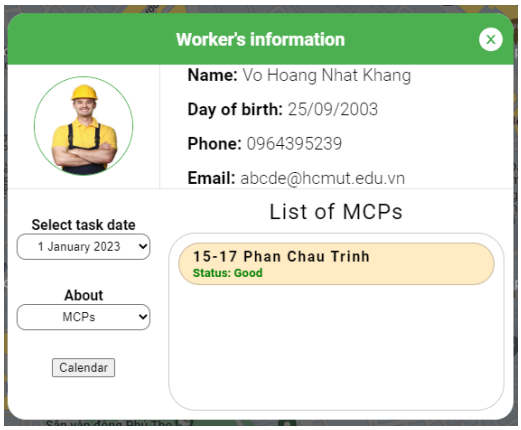



**Figure 5.5:** Worker's personal information

Here, Back officers can view almost anything: Name, Day of birth, Phone... or they can even display task date of the worker, or even display progressing tasks.

### 5.1.4 View MCP on map

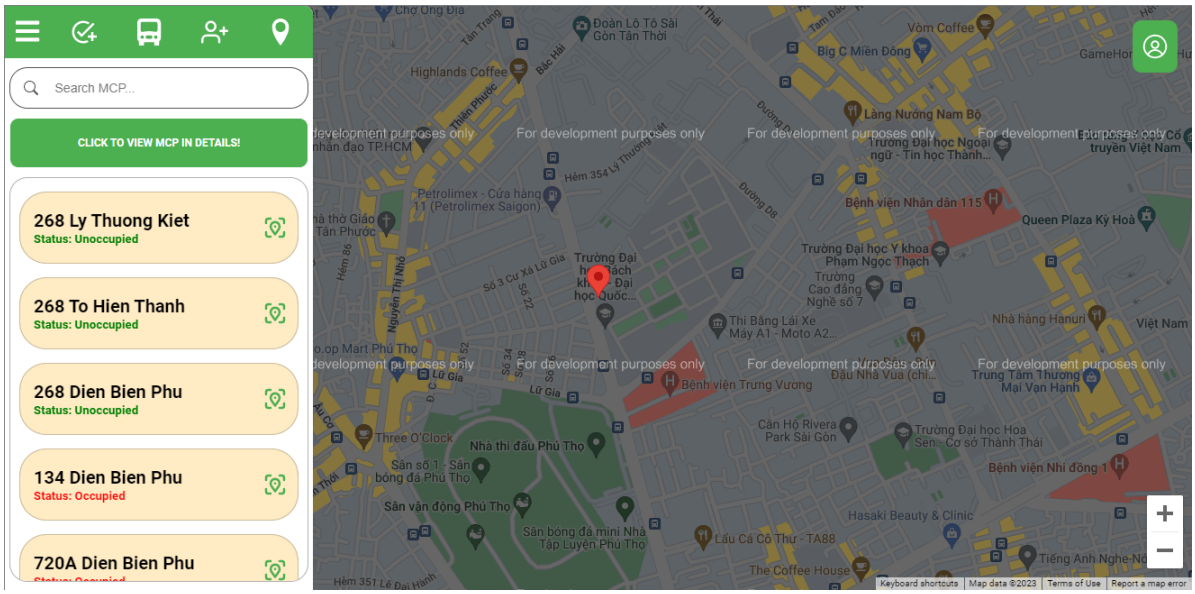



**Figure 5.6:** MCPs on map

We also added additional features, including moving to the MCP on map once user click on the icon (Interactive map)

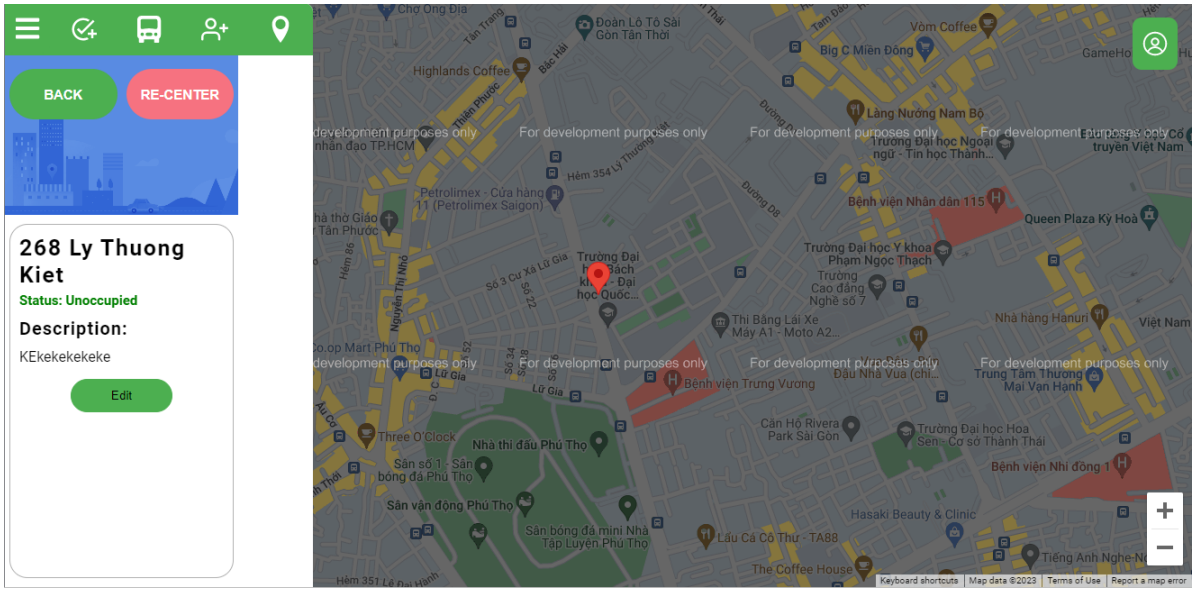



**Figure 5.7:** MCP information

## 5.2 Demonstrate the whole project from Task 1 to Task 5

For better demonstration, we provided you with some slides. Slide link: https://1drv.ms/p/s!AgVTvYPl5Zm1jUZcdO1njTQatxu1?e=Aze7At

But we will walk through this project once again, to have a review about our designs, models, wireframe and implementations.

### 5.2.1 Key designs

There are two main use-case diagram in this project, that is the system use-case and task management (most important) use-case diagram. Description of diagrams can be read again at Task 1.

### 5.2.2 Key models

For our models, there are 4 main diagrams:

- **Activity diagram:** Activity diagrams help people on the business and development sides of an organization come together to understand the same process and behavior. In our project, the activity diagram demonstrated the business process between systems and the actors or stakeholders taken part in the whole system.
- **Sequence diagram:** This showed how back officers can assign vehicles to the workers. This also show how operations are carried out inside a system, especially the interactions between the objects. In our system, the vehicle assignment module has the most importing job: Keeping the connection between the Database and the Interface alive, as well as retrieving required data and also receiving requests from Interface.
- **Class diagram:** This diagram demonstrate clearly how system is structured between classes, methods, and the relationship among them. Our system consist of 5 layers, except the fact that the Service layers will externalize the APIs from Business layers, which in fact my reduce the number of layers to 4.
- **Component diagram:** Based on the layered architecture, we continue visualizing how components in a layer are organized. Furthermore, we need to implicitly define how these components interact which each others throughout layers.

The above represented diagrams can be reviewed at Task 2.

### 5.2.3 Wireframes

In this project we chose Figma for showing some basic UIs as well as the main features in out application. Especially we also use Figma for pointing out how can routes be seen on the map:

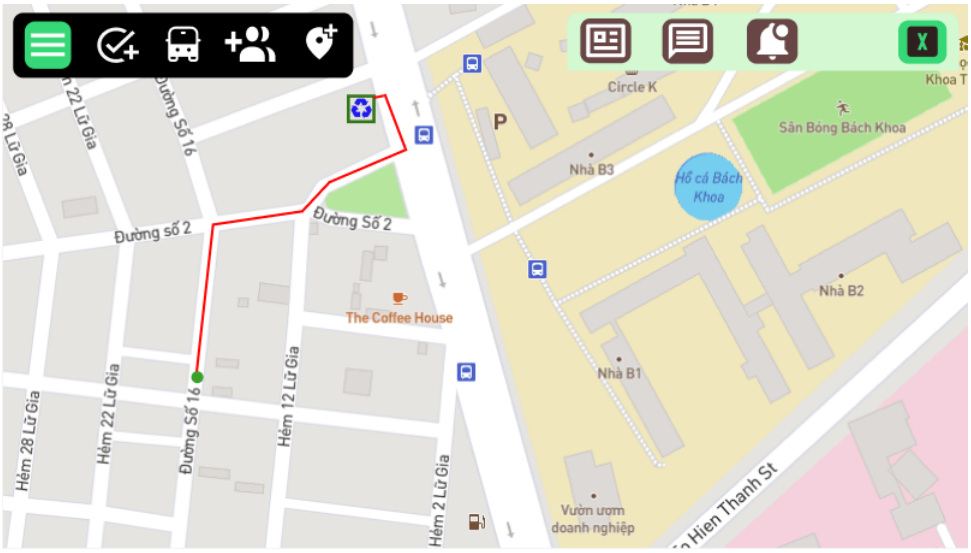



**Figure 5.8:** Routes on map

Users can also check for MCP status by clicking on the MCP:


**Figure 5.9:** Live MCP information

More UIs and wireframe explanations can be viewed from Task 2.
The development of MVCs can be divided into 3 phases in this project:

- **Phase 1:** Is when we first design our very first UIs with Figma (MVC1).
- **Phase 2:** This is the time we send out our demo to users who are interested in using the application as well as stakeholders (in this case, the stakeholders will be our customers) (MVC2).
- **Phase 3:** After receiving valuable feedbacks, we will restructure the wireframes, UIs, improving UX. Next is to code them up (MVC3).

### 5.2.4 Implementations

Implementation is how we structured our code. We tried to design the codes based on the logic that objects communicate together like components interacts together in the system. Therefore, the way we manage our folders, directories and how we handle exceptions will have a big effect at our application. See this at Task 4.

# Chapter 6

# Conclusion and some opinions

After finishing UWC 2.0 application, our team have a lot of thoughts:

- **Satisfaction:** The feeling of satisfaction after successfully building a web application that meets the requirements can be immense. It is a testament to the hard work and effort put in by the team and can be a source of pride.
- **Relief:** Building a web application is a complex and challenging process that requires a lot of time and effort. Once we have finished the application, there is a sense of relief that the project is complete and that we can move onto other projects.
- **Reflection:** After finishing a web application, it's essential to reflect on the development process and overall project. This reflection can help to identify areas of improvement and learnings, allowing for continuous development and improvement.
- **Feedback:** Getting feedback from users or stakeholders after completing a web application can be valuable. It can help to identify areas of the application that require further development, as well as highlight areas where the application excels.
- **Excitement:** Having finished a web application can be an exciting time. It opens up the possibility of launching the application and seeing how it performs in real-world scenarios.